Beams
Department

Methodical Accelerator Design

Overview of 'Next Generation'

FCCIS WP2 - CERN.

Laurent Deniau

CERN-BE/ABP

1st December 2021

MAD-NG objectives

- **Long term design: easy to use and extend.**
  - ➡ Flexible language ➠ **fast, simple, and general purpose scripting language.**
    - ~70% of the code is written in the scripting language, ~30% in C.
  - ➡ Flexible technologies ➠ **self-contained, all-in-one and modular.**
    - single application, no dependencies (except Gnuplot for plotting).
  - ➡ Efficient & Portable technologies ➠ **embeds a Just in Time compiler.**
    - same results everywhere (LNX, OSX, WIN), extensive unit tests (>8000).
    - fast and extremely simple Foreign Function Interface to C, C++, Fortran, etc…
- **6D PTC physics using GTPSA (for DA) and symplectic integrators.**
    - slicing, combined physics, combined elements, support/development for extensions is easy…
- **Development open source.**
  - ➡ GitHub https://github.com/MethodicalAcceleratorDesign/MAD
  - ➡ License GPL V3, User manual (~180p, covers <20%), Programmer Manual (29p).

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG objectives

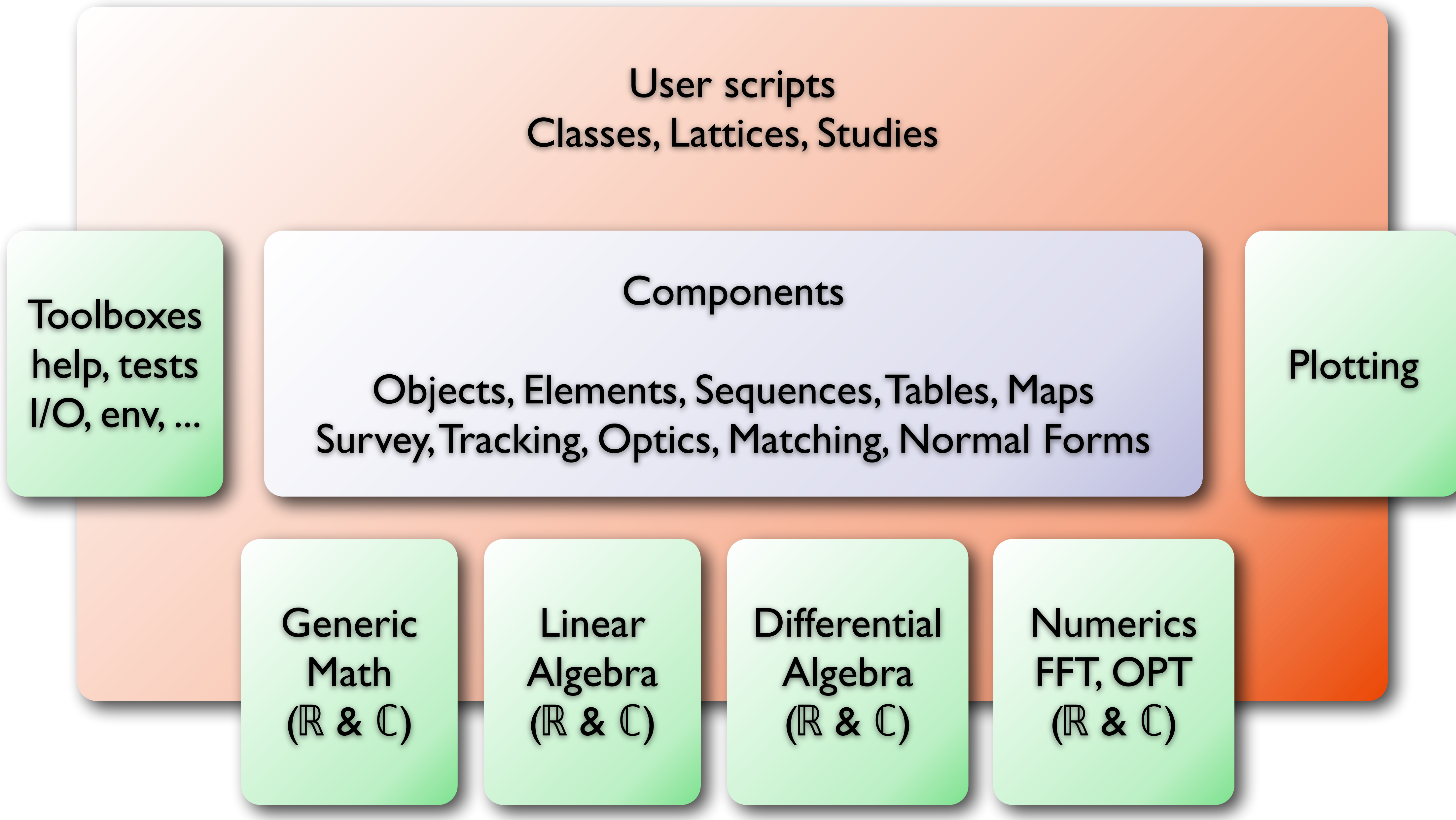
Beams Department

- **Long term design: easy to use and extend.**
  - ➡ Flexible language ➠ **fast, simple, and general purpose scripting language.**
    - ~70% of the code is written in the scripting language, ~30% in C.
  - ➡ Flexible technologies ➠ **self-contained, all-in-one and modular.**
    - single application, no dependencies (except Gnuplot for plotting).
  - ➡ Efficient & Portable technologies ➠ **embeds a Just in Time compiler.**
    - same results everywhere (LNX, OSX, WIN), extensive unit tests (>8000).
    - fast and extremely simple Foreign Function Interface to C, C++, Fortran, etc…
- **6D PTC physics using GTPSA (for DA) and symplectic integrators.**
    - slicing, combined physics, combined elements, support/development for extensions is easy…
- **Development open source.**
  - ➡ GitHub https://github.com/MethodicalAcceleratorDesign/MAD
  - ➡ License GPL V3, User manual (~180p, covers <20%), Programmer Manual (29p).

- intended objective -
provide a general platform to
develop tracking and optics codes
for accelerator beam physics.

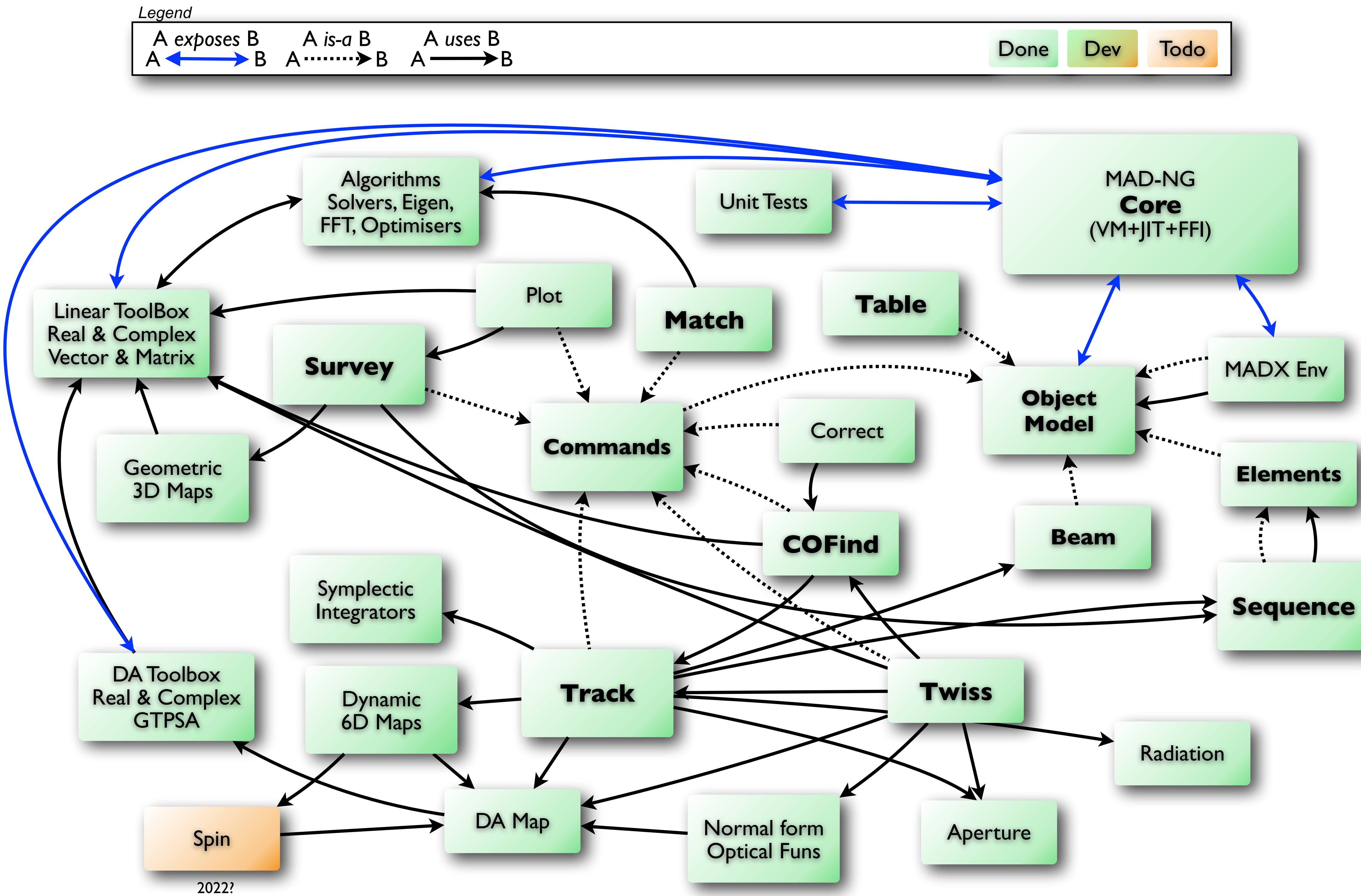
MAD-NG schematic layout

- ***Built from the start as a platform to develop & benchmark physics.***
  - ➡ *Everything is accessible, modifiable and extensible **by users from scripts** (e.g. even at runtime).*

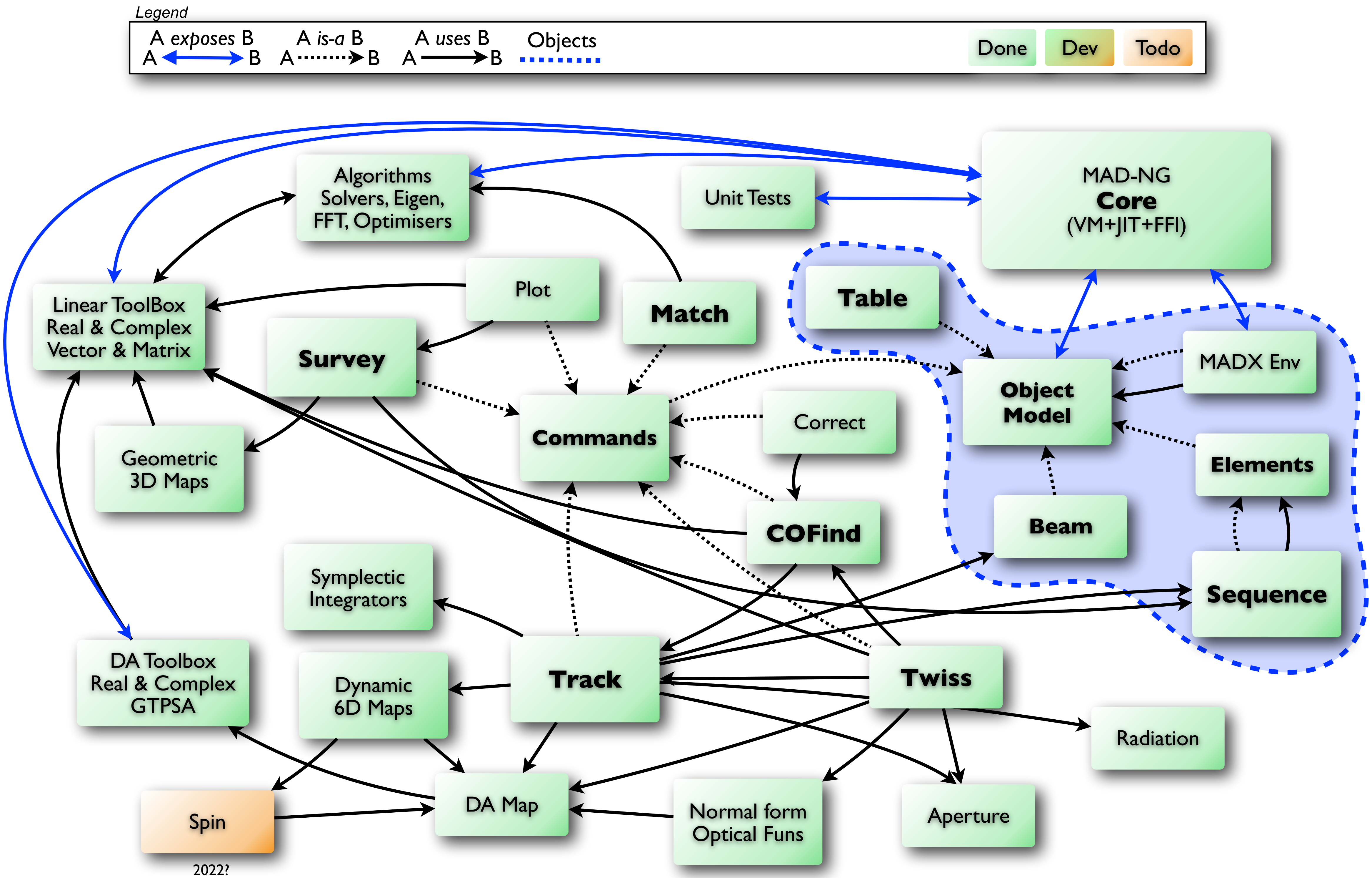


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

MAD-NG ecosystem

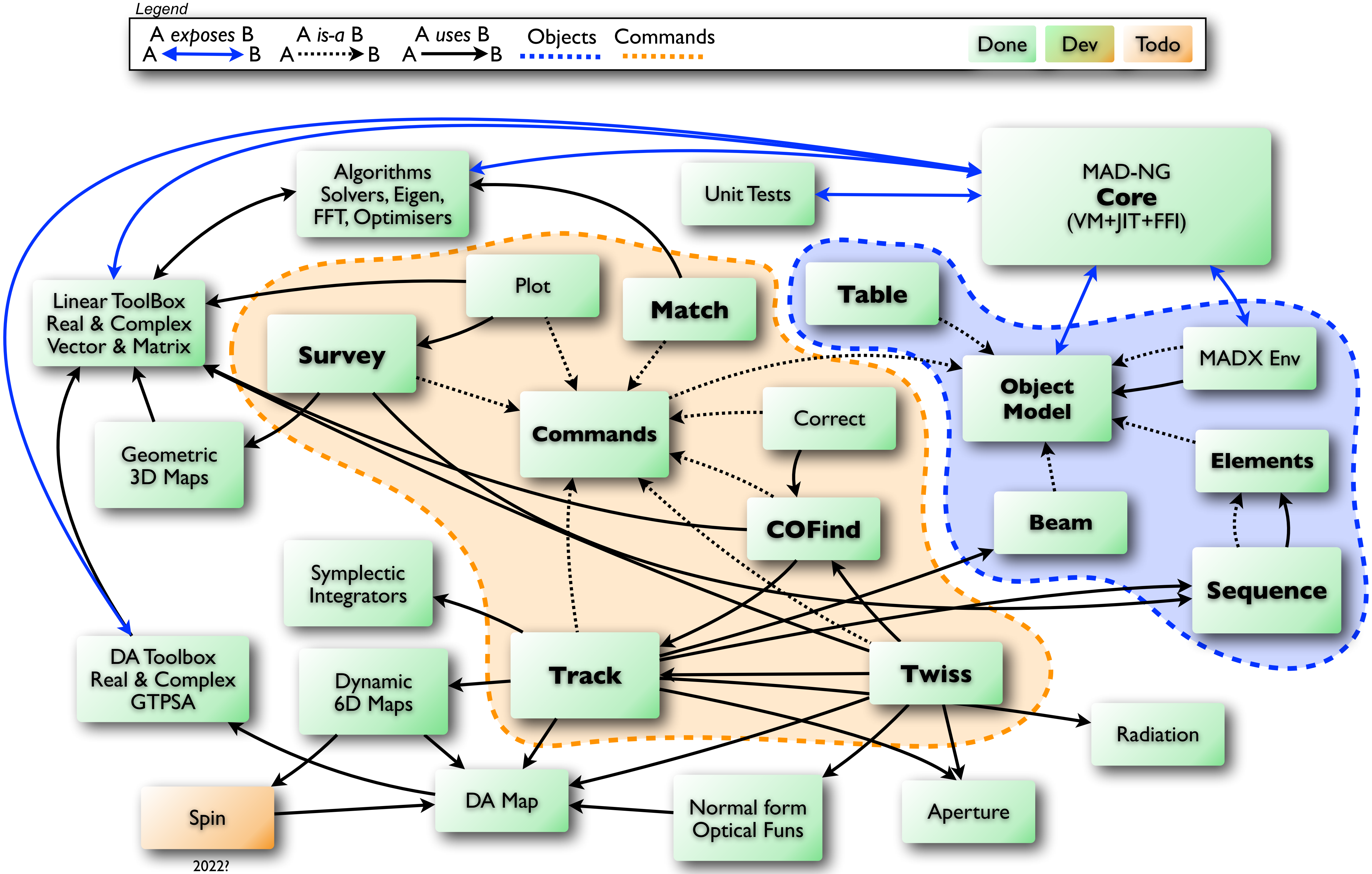


MAD-NG ecosystem

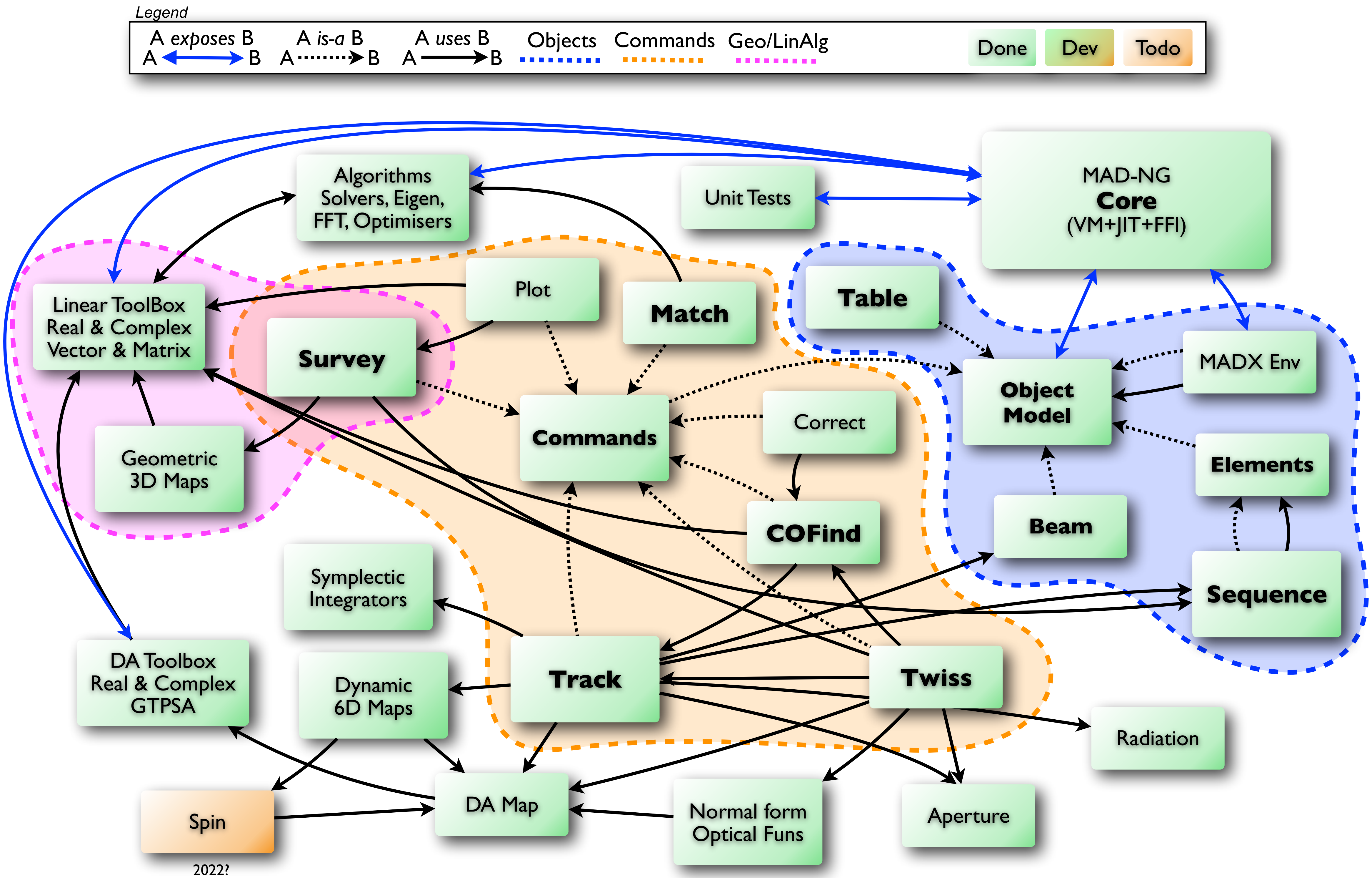


Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG ecosystem

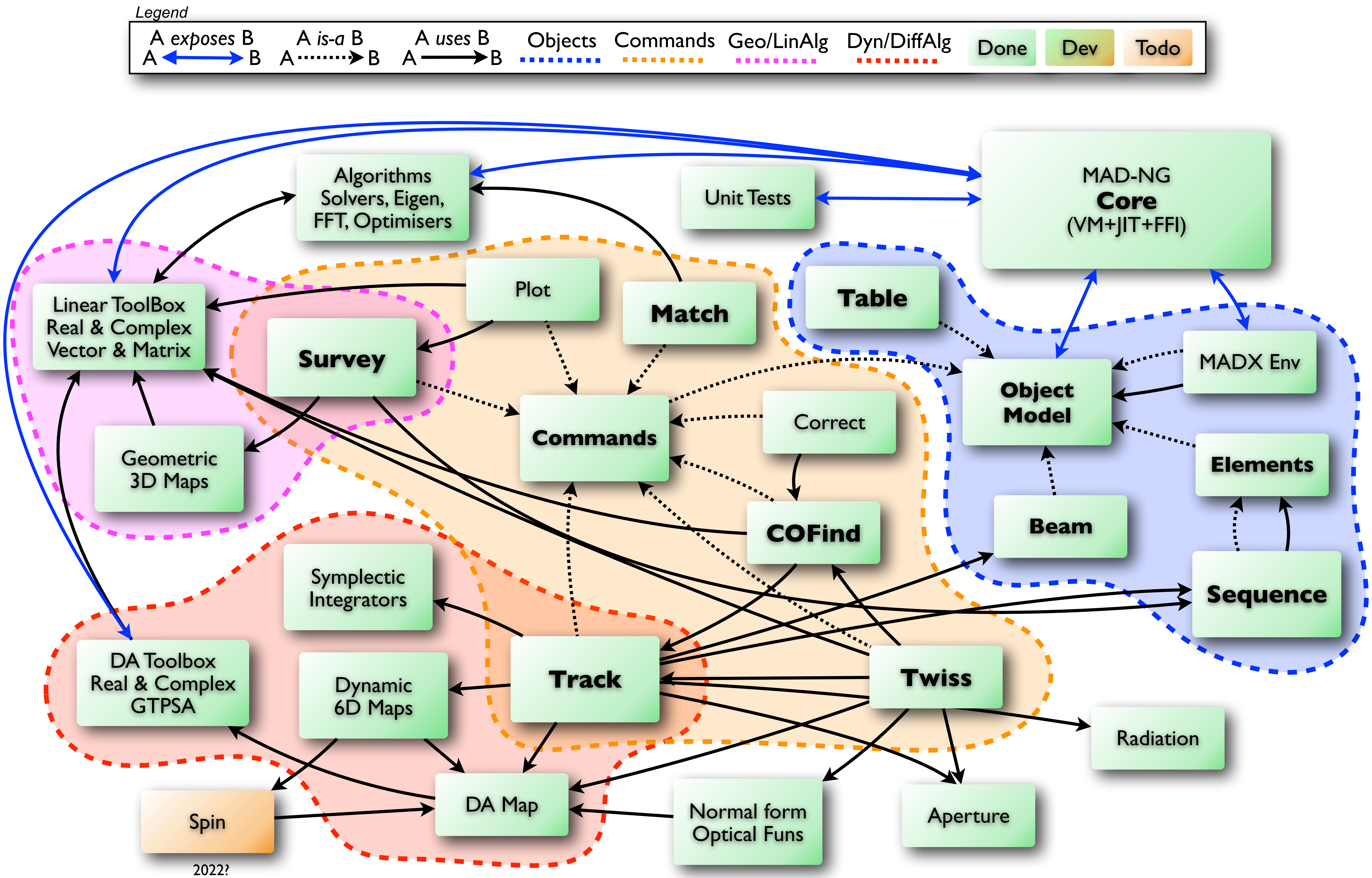


MAD-NG ecosystem



Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG ecosystem



Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Sequences & elements

SPS in MAD-X

```
SPS:    LINE = (6*SUPER);
SUPER:  LINE = (7*P44,INSERT,7*P44);
INSERT: LINE = (P24,2*P00,P42);
P00:    LINE = (QF,DL,QD,DL);
P24:    LINE = (QF,DM,2*B2,DS,PD);
P42:    LINE = (PF,QD,2*B2,DM,DS);
P44:    LINE = (PF,PD);
PD:     LINE = (QD,2*B2,2*B1,DS);
PF:     LINE = (QF,2*B1,2*B2,DS);
```

SPS in MAD-NG

```
pf     = bline {qf,2*b1,2*b2,ds}
pd     = bline {qd,2*b2,2*b1,ds}
p24    = bline {qf,dm,2*b2,ds,pd}
p42    = bline {pf,qd,2*b2,dm,ds}
p00    = bline {qf,dl,qd,dl}
p44    = bline {pf,pd}
insert = bline {p24,2*p00,p42}
super  = bline {7*p44,insert,7*p44}
SPS    = sequence 'SPS' {6*super}
```

- **Lattices definition as in MAD-X** *(syntax is very close)*
  - ➡ sequences are both containers (e.g. access elements) and table (store arbitrary objects).
    - e.g. to store their **beam** or their own list of **knobs**.
  - ➡ elements are both containers (e.g. access attributes) and table (store arbitrary objects).
  - ➡ sequence can include sub**sequence**s, beam **line**s and **element**s (and sub**element**s).
  - ➡ operator overloading (+, -, *) allows to mix *lines* and *sequences* descriptions arbitrarily.
  - ➡ names are optional and can be non-unique with support for *relative* or *absolute* counts.
    - positions 'AT' can be absolute or relative 'FROM' names with absolute or relative counts.

SPS in MAD-X

```
SPS:     LINE = (6*SUPER);
SUPER:   LINE = (7*P44,INSERT,7*P44);
INSERT:  LINE = (P24,2*P00,P42);
P00:     LINE = (QF,DL,QD,DL);
P24:     LINE = (QF,DM,2*B2,DS,PD);
P42:     LINE = (PF,QD,2*B2,DM,DS);
P44:     LINE = (PF,PD);
PD:      LINE = (QD,2*B2,2*B1,DS);
PF:      LINE = (QF,2*B1,2*B2,DS);
```

SPS in MAD-NG

```
pf     = bline {qf,2*b1,2*b2,ds}
pd     = bline {qd,2*b2,2*b1,ds}
p24    = bline {qf,dm,2*b2,ds,pd}
p42    = bline {pf,qd,2*b2,dm,ds}
p00    = bline {qf,dl,qd,dl}
p44    = bline {pf,pd}
insert = bline {p24,2*p00,p42}
super  = bline {7*p44,insert,7*p44}
SPS    = sequence 'SPS' {6*super}
```


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

- **Lattices definition as in MAD-X** *(syntax is very close)*
  - ➡ sequences are both containers (e.g. access elements) and table
    - e.g. to store their **beam** or their own list of **knobs**.
  - ➡ elements are both containers (e.g. access attributes) and table (store arbitrary objects).
  - ➡ sequence can include sub**sequence**s, beam **line**s and **element**s (and sub**elements**).
  - ➡ operator overloading (+, -, *) allows to mix *lines* and *sequences* descriptions arbitrarily.
  - ➡ names are optional and can be non-unique with support for *relative* or *absolute* counts.
    - positions 'AT' can be absolute or relative 'FROM' names with absolute or relative counts.

unified definitions of lines and sequences plus extensions

SPS in MAD-X

```
SPS:    LINE = (6*SUPER);
SUPER:  LINE = (7*P44,INSERT,7*P44);
INSERT: LINE = (P24,2*P00,P42);
P00:    LINE = (QF,DL,QD,DL);
P24:    LINE = (QF,DM,2*B2,DS,PD);
P42:    LINE = (PF,QD,2*B2,DM,DS);
P44:    LINE = (PF,PD);
PD:     LINE = (QD,2*B2,2*B1,DS);
PF:     LINE = (QF,2*B1,2*B2,DS);
```

SPS in MAD-NG

```
pf     = bline {qf,2*b1,2*b2,ds}
pd     = bline {qd,2*b2,2*b1,ds}
p24    = bline {qf,dm,2*b2,ds,pd}
p42    = bline {pf,qd,2*b2,dm,ds}
p00    = bline {qf,dl,qd,dl}
p44    = bline {pf,pd}
insert = bline {p24,2*p00,p42}
super  = bline {7*p44,insert,7*p44}
SPS    = sequence 'SPS' {6*super}
```

Sequences & elements

- **Lattices definition as in MAD-X** *(syntax is very close)*
  - ➡ sequences are both containers (e.g. access elements) and table
    - e.g. to store their **beam** or their own list of **knobs**.
  - ➡ elements are both containers (e.g. access attributes) and table (store arbitrary objects).
  - ➡ sequence can include sub**sequence**s, beam **line**s and **element**s (and sub**element**s).
  - ➡ operator overloading (+, -, *) allows to mix *lines* and *sequences* descriptions arbitrarily.
  - ➡ names are optional and can be non-unique with support for *relative* or *absolute* counts.
    - positions ‘AT' can be absolute or relative ‘FROM’ names with absolute or relative counts.

unified definitions of lines and sequences plus extensions

- Manage arbitrary number of sequences to allow simulation of **entire accelerators complex**.
  - ➡ Shared sequences, e.g. LHCB1 and LHCB2.
    - provides few sharing policies.
  - ➡ Chained sequences, e.g. PSB, PS, SPS and BTL.
  - ➡ Conditionally chained sequences (e.g. RaceTrack).
    - e.g. Booster ➠ Ring1 if energy > 175 GeV
    - based on special *s-link* element
    - connections and conditions are performed through an arbitrary user-defined function.

SPS in MAD-X

```
SPS:    LINE = (6*SUPER);
SUPER:  LINE = (7*P44,INSERT,7*P44);
INSERT: LINE = (P24,2*P00,P42);
P00:    LINE = (QF,DL,QD,DL);
P24:    LINE = (QF,DM,2*B2,DS,PD);
P42:    LINE = (PF,QD,2*B2,DM,DS);
P44:    LINE = (PF,PD);
PD:     LINE = (QD,2*B2,2*B1,DS);
PF:     LINE = (QF,2*B1,2*B2,DS);
```

SPS in MAD-NG

```
pf     = bline {qf,2*b1,2*b2,ds}
pd     = bline {qd,2*b2,2*b1,ds}
p24    = bline {qf,dm,2*b2,ds,pd}
p42    = bline {pf,qd,2*b2,dm,ds}
p00    = bline {qf,dl,qd,dl}
p44    = bline {pf,pd}
insert = bline {p24,2*p00,p42}
super  = bline {7*p44,insert,7*p44}
SPS    = sequence 'SPS' {6*super}
```


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Sequences conversion (MAD-X to MAD)

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Sequences conversion (MAD-X to MAD)

- MAD-NG **loads and convert** MAD-X sequences, elements and variables, *including deferred expressions,* **on-the-fly** into the MADX environment (a MAD-NG context that emulates MAD-X global workspace) and/or save conversion to files.

```
! convert MAD-X files on need, save to MAD file (disk), load to MADX environment (memory)
  MADX:load('lhc_as-built.seq'        , 'lhc_as-built.mad')
  MADX:load('opticsfile.22_ctpps2'    , 'opticsfile.22_ctpps2.mad')
  MADX:load("FCCee_z_213_nosol_18.seq", "FCCee_z_213_nosol_18.mad")
```

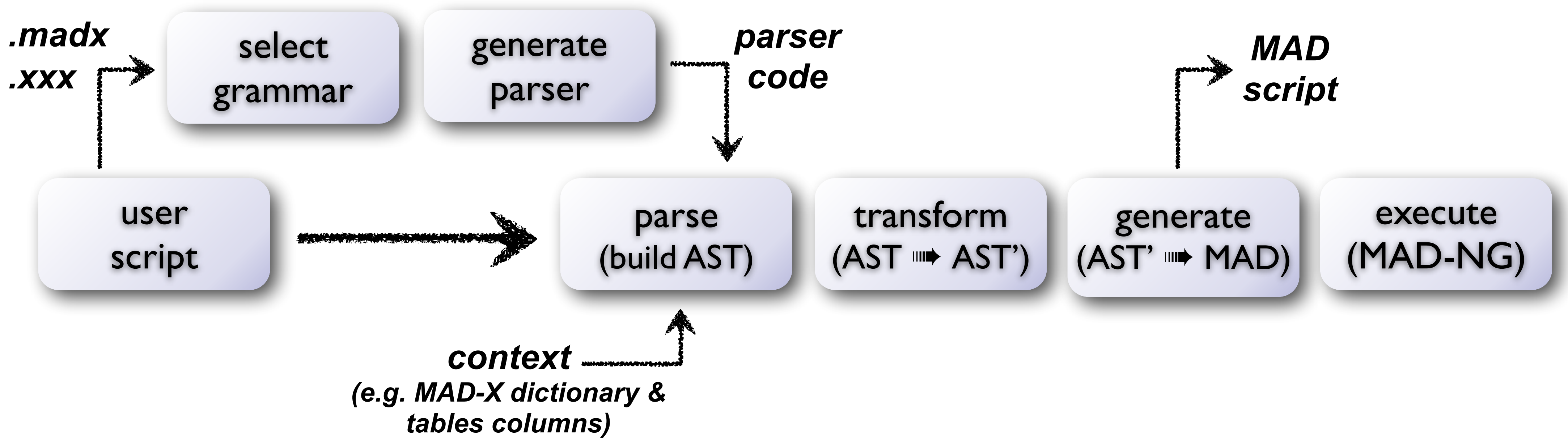
Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Sequences conversion (MAD-X to MAD)

- MAD-NG **loads and convert** MAD-X sequences, elements and variables, *including deferred expressions,* **on-the-fly** into the MADX environment (a MAD-NG context that emulates MAD-X global workspace) and/or save conversion to files.

```
! convert MAD-X files on need, save to MAD file (disk), load to MADX environment (memory)
  MADX:load('lhc_as-built.seq'         , 'lhc_as-built.mad')
  MADX:load(‘opticsfile.22_ctpps2'     , ‘opticsfile.22_ctpps2.mad')
  MADX:load("FCCee_z_213_nosol_18.seq", "FCCee_z_213_nosol_18.mad")
```

- MAD-NG embeds technologies to **parse arbitrary language** that can be **described with PEG** (parser expression grammar) to generate AST (abstract syntax tree), and apply transformations and/or evaluations.



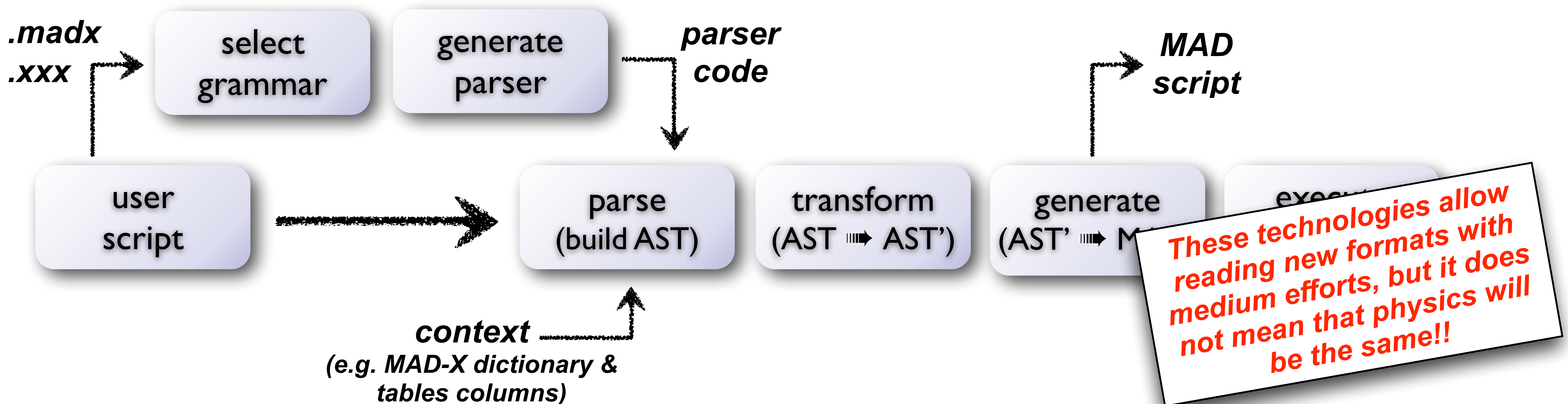
Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Sequences conversion (MAD-X to MAD)

- MAD-NG **loads and convert** MAD-X sequences, elements and variables, *including deferred expressions*, **on-the-fly** into the MADX environment (a MAD-NG context that emulates MAD-X global workspace) and/or save conversion to files.

```
! convert MAD-X files on need, save to MAD file (disk), load to MADX environment (memory)
  MADX:load('lhc_as-built.seq'        , 'lhc_as-built.mad')
  MADX:load('opticsfile.22_ctpps2'    , 'opticsfile.22_ctpps2.mad')
  MADX:load("FCCee_z_213_nosol_18.seq", "FCCee_z_213_nosol_18.mad")
```

- MAD-NG embeds technologies to **parse arbitrary language** that can be **described with PEG** (parser expression grammar) to generate AST (abstract syntax tree), and apply transformations and/or evaluations.

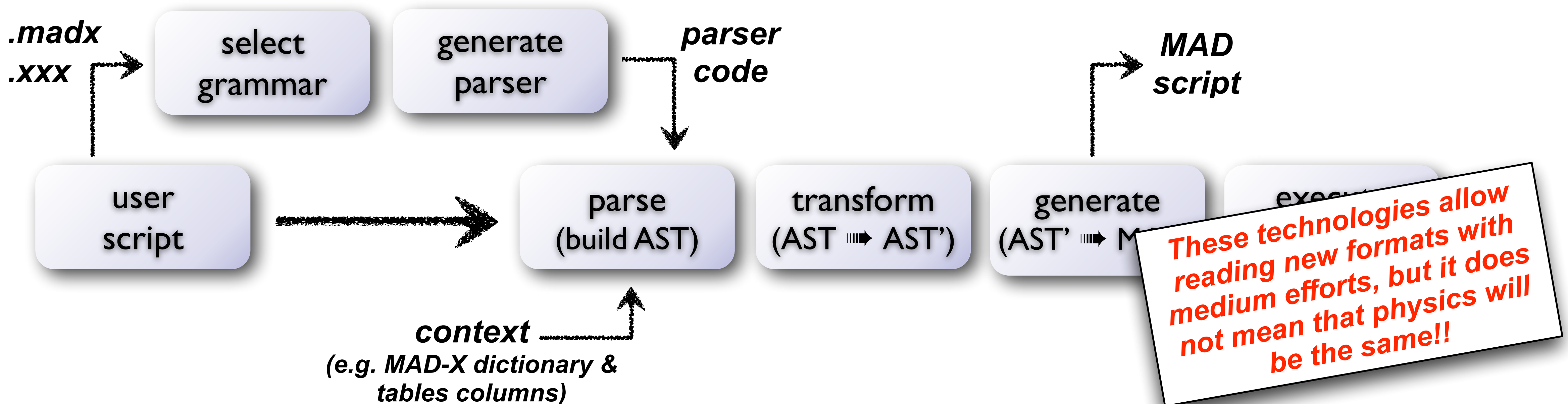


Sequences conversion (MAD-X to MAD)

- MAD-NG **loads and convert** MAD-X sequences, elements and variables, *including deferred expressions,* **on-the-fly** into the MADX environment (a MAD-NG context that emulates MAD-X global workspace) and/or save conversion to files.

```
! convert MAD-X files on need, save to MAD file (disk), load to MADX environment (memory)
  MADX:load('lhc_as-built.seq'          , 'lhc_as-built.mad')
  MADX:load('opticsfile.22_ctpps2'      , 'opticsfile.22_ctpps2.mad')
  MADX:load("FCCee_z_213_nosol_18.seq", "FCCee_z_213_nosol_18.mad")
```

- MAD-NG embeds technologies to **parse arbitrary language** that can be **described with PEG** (parser expression grammar) to generate AST (abstract syntax tree), and apply transformations and/or evaluations.



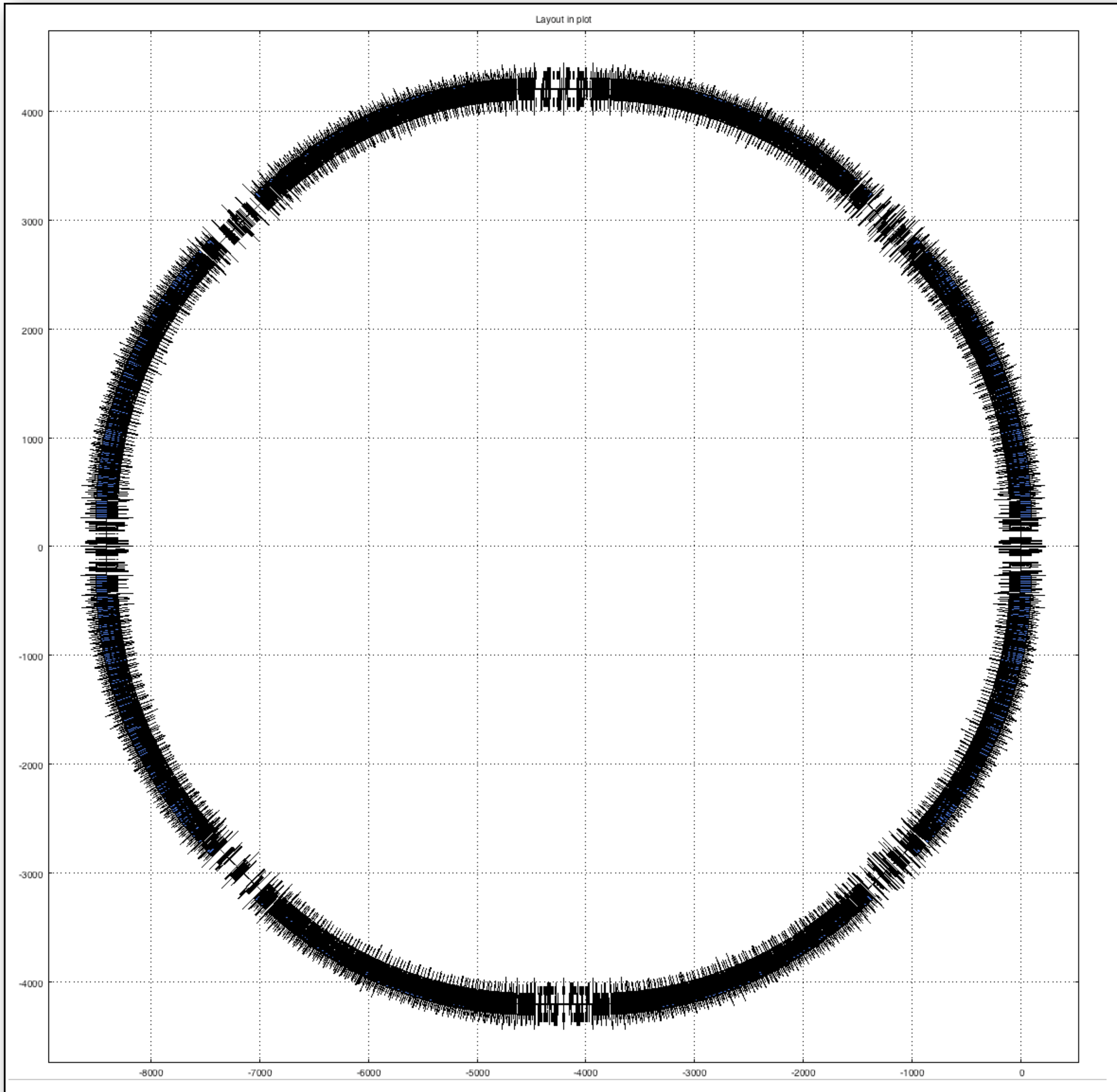
- MAD-NG allows to run MAD-X as a module to convert sequences, elements and variables into MADX environment as with CpyMad. But this method does not propagate the deferred expressions, i.e. lattice logic is lost (fine for a “static” description). Could be propagated with some extra work.

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Sequence plot *(LHC 1 & 2 survey)*

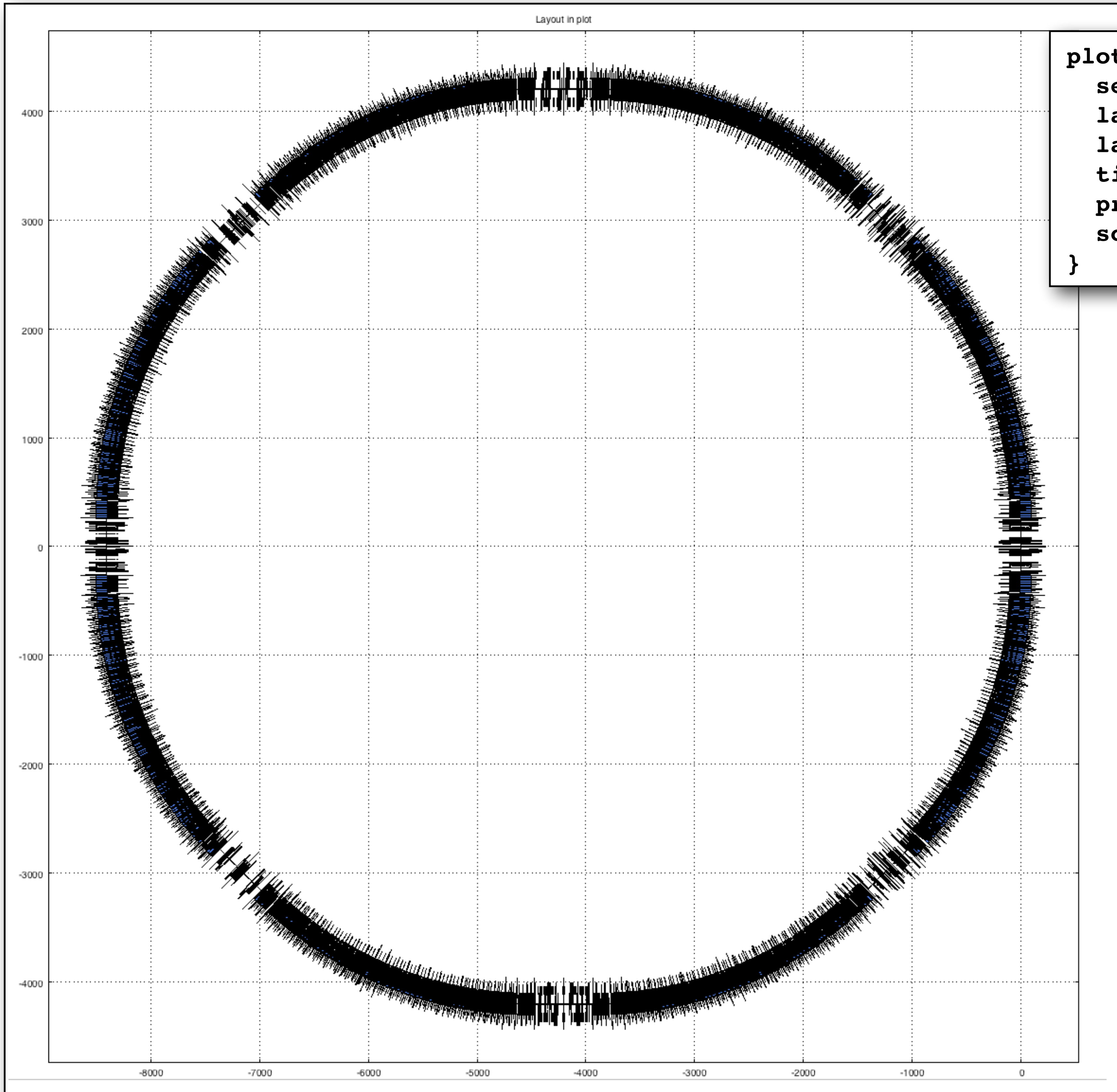
Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Sequence plot *(LHC 1 & 2 survey)*



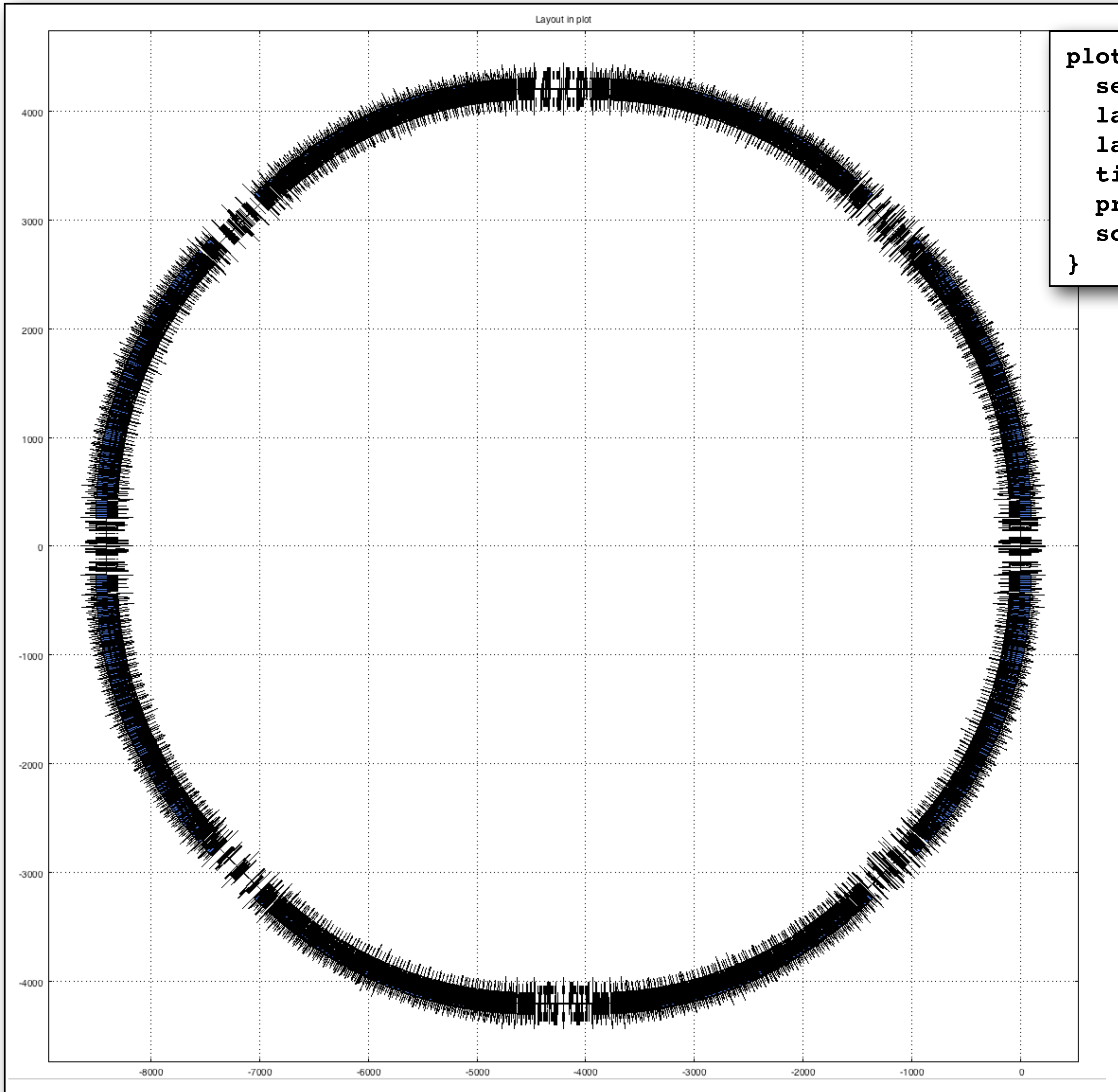
Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Sequence plot *(LHC 1 & 2 survey)*




```
plot {
  sequence = {lhcb1,lhcb2},
  laypos   = "in",
  layonly  = false,
  title    = "Layout in plot",
  prolog   = 'set size ratio -1',
  scrdump  = "plotlhc.gp",
}
```

Sequence plot *(LHC 1 & 2 survey)*




```
plot {
  sequence = {lhcb1,lhcb2},
  laypos   = "in",
  layonly  = false,
  title    = "Layout in plot",
  prolog   = 'set size ratio -1',
  scrdump  = "plotlhc.gp",
}
```

MAD-X loads the entire LHC definition in ~1 s. (2 beamlines, ~30000 lines)

MAD-NG loads the entire LHC in MAD-X format and saved it in files in ~1 s.

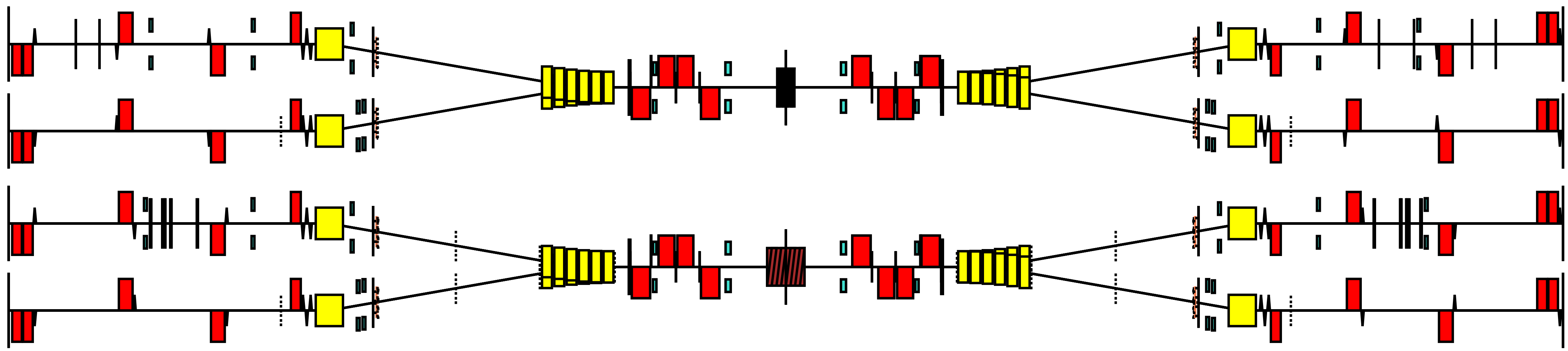
MAD-NG loads the entire LHC from converted files (.mad files) in ~0.2 s.

Gnuplot script (.gp files) size is 5 MB & 125000+ lines and takes ~1 sec to display.

All items are tagged i.e. moving the mouse over show their name and kind

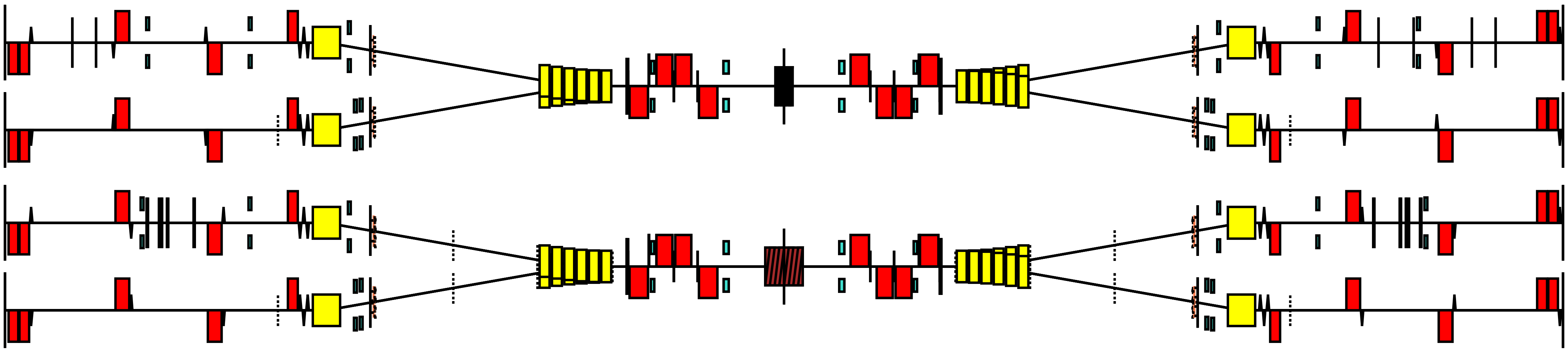
Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Sequence plot *(LHC 1 & 2 at IP1 & IP5 layout)*



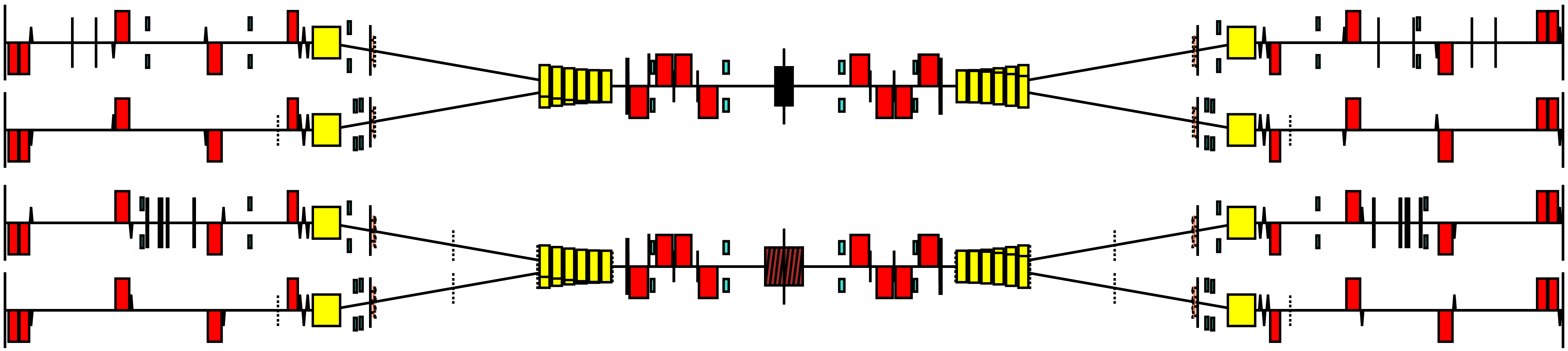
Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Sequence plot *(LHC 1 & 2 at IP1 & IP5 layout)*



```
plot {
  sequence = { lhcb1, lhcb2, lhcb1, lhcb2 },
  range    = {
    {"E.DS.L1.B1","S.DS.R1.B1"},{"E.DS.L1.B2","S.DS.R1.B2"},
    {"E.DS.L5.B1","S.DS.R5.B1"},{"E.DS.L5.B2","S.DS.R5.B2"},
  },
  laydisty = {
    lhcb2["E.DS.L1.B2"].mech_sep,        ! second bline
    -0.4,                                ! third  bline
    -0.4 + lhcb2['E.DS.L5.B2'].mech_sep ! fourth bline
  },
  title = "IP1-IP5 two angled beams",
}
```

Sequence plot *(LHC 1 & 2 at IP1 & IP5 layout)*



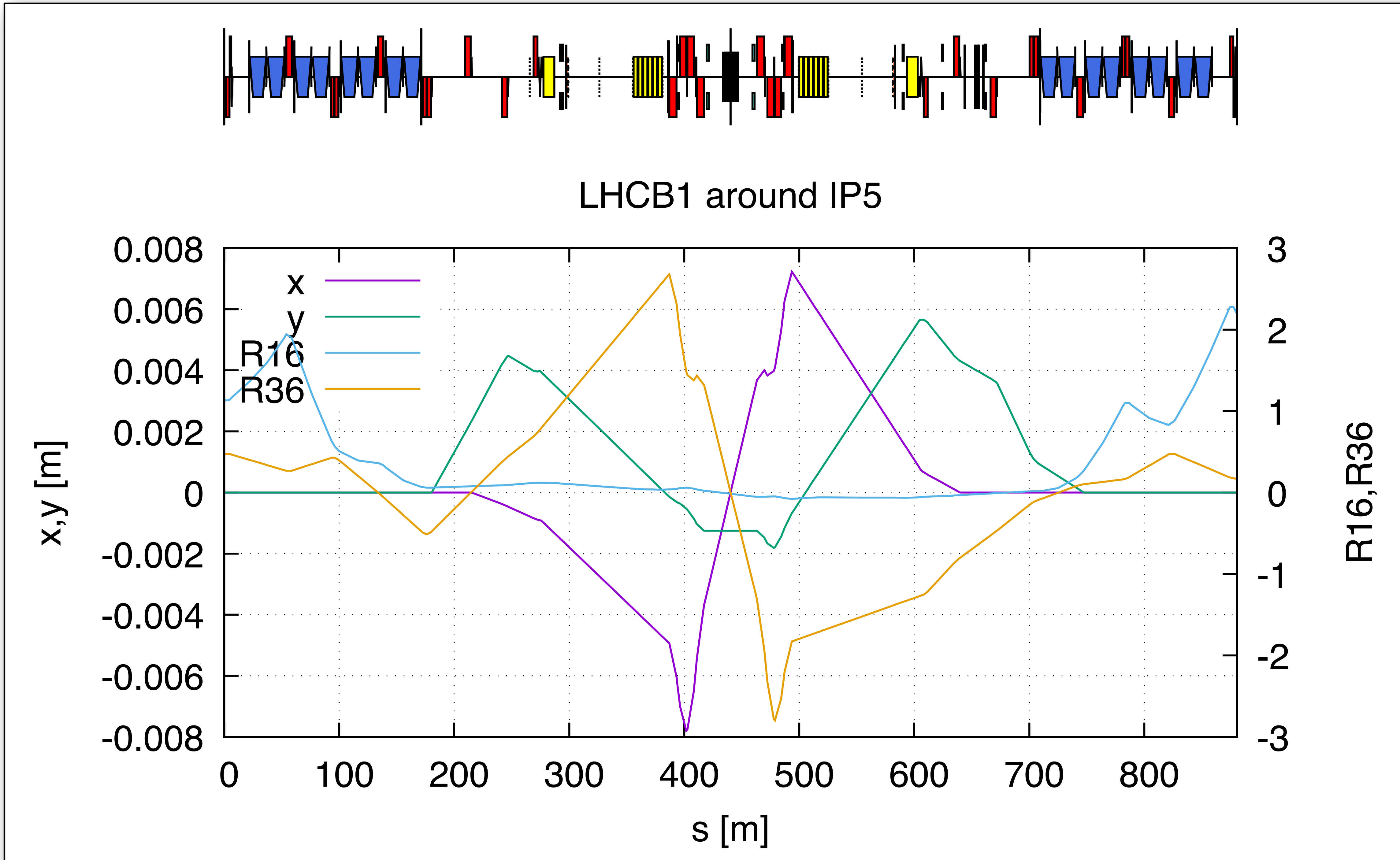
```
plot {
  sequence = { lhcb1, lhcb2, lhcb1, lhcb2 },
  range    = {
    {“E.DS.L1.B1","S.DS.R1.B1"},{"E.DS.L1.B2","S.DS.R1.B2"},
    {"E.DS.L5.B1","S.DS.R5.B1"},{"E.DS.L5.B2","S.DS.R5.B2"},
  },
  laydisty = {
    lhcb2[“E.DS.L1.B2"].mech_sep,       ! second bline
    -0.4,                               ! third  bline
    -0.4 + lhcb2[‘E.DS.L5.B2'].mech_sep ! fourth bline
  },
  title = "IP1-IP5 two angled beams",
}
```

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

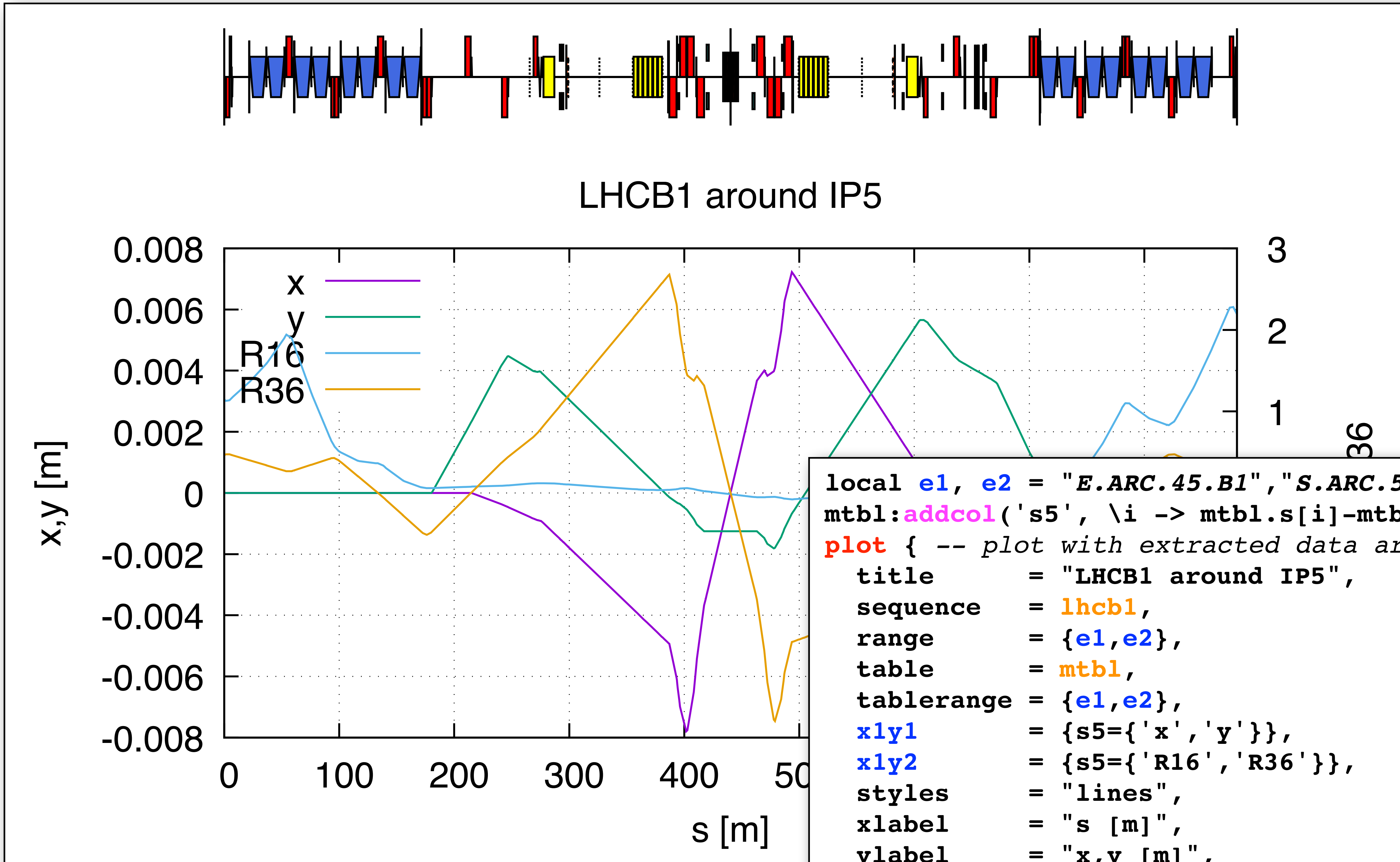
Track plot (LHCB1 around IP5)

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Track plot (LHCB1 around IP5)



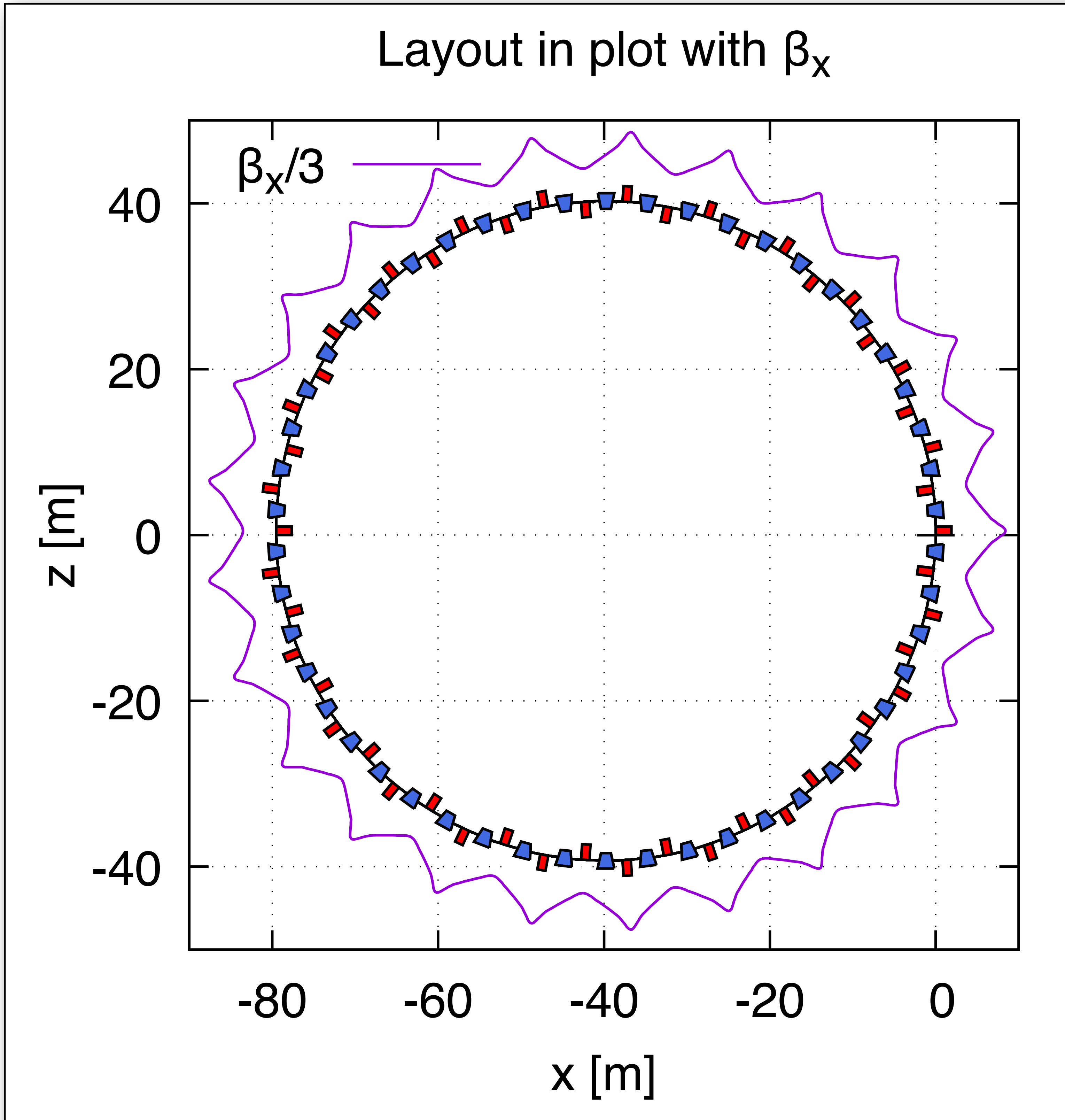
Track plot (LHCB1 around IP5)



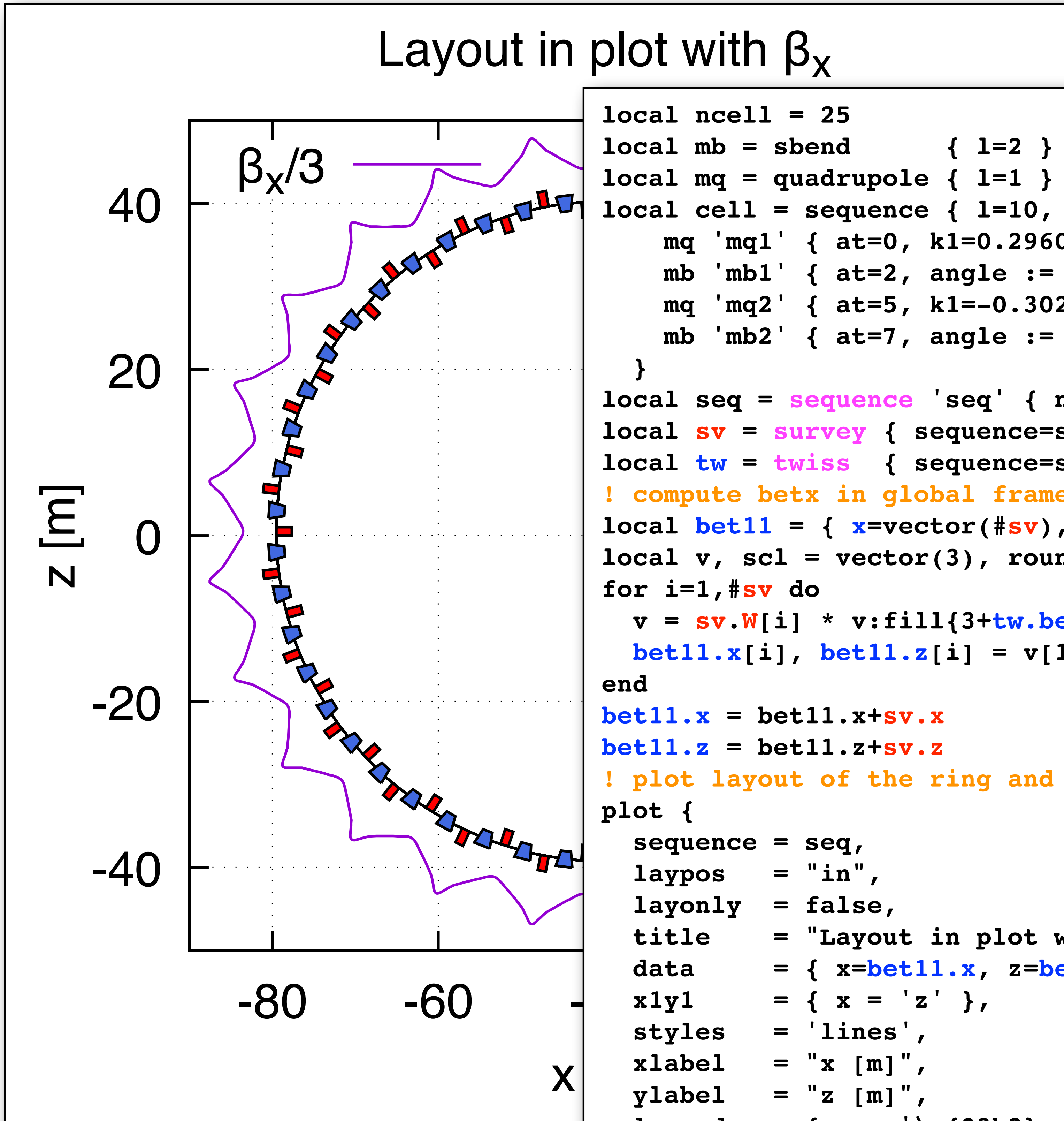

```
local e1, e2 = "E.ARC.45.B1","S.ARC.56.B1"
mtbl:addcol('s5', \i -> mtbl.s[i]-mtbl[e1].s)
plot { -- plot with extracted data around IP5
  title      = "LHCB1 around IP5",
  sequence   = lhcb1,
  range      = {e1,e2},
  table      = mtbl,
  tablerange = {e1,e2},
  x1y1       = {s5={'x','y'}},
  x1y2       = {s5={'R16','R36'}},
  styles     = "lines",
  xlabel     = "s [m]",
  ylabel     = "x,y [m]",
  y2label    = "R16,R36",
  fontsize   = 14,
  output     = “plots/orbit_lhcb1_ip5_da.pdf",
--scrdump    = "plots/orbit_lhcb1_ip5.gp",
}
```

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Plot survey & twiss *(two rings with β_x)*

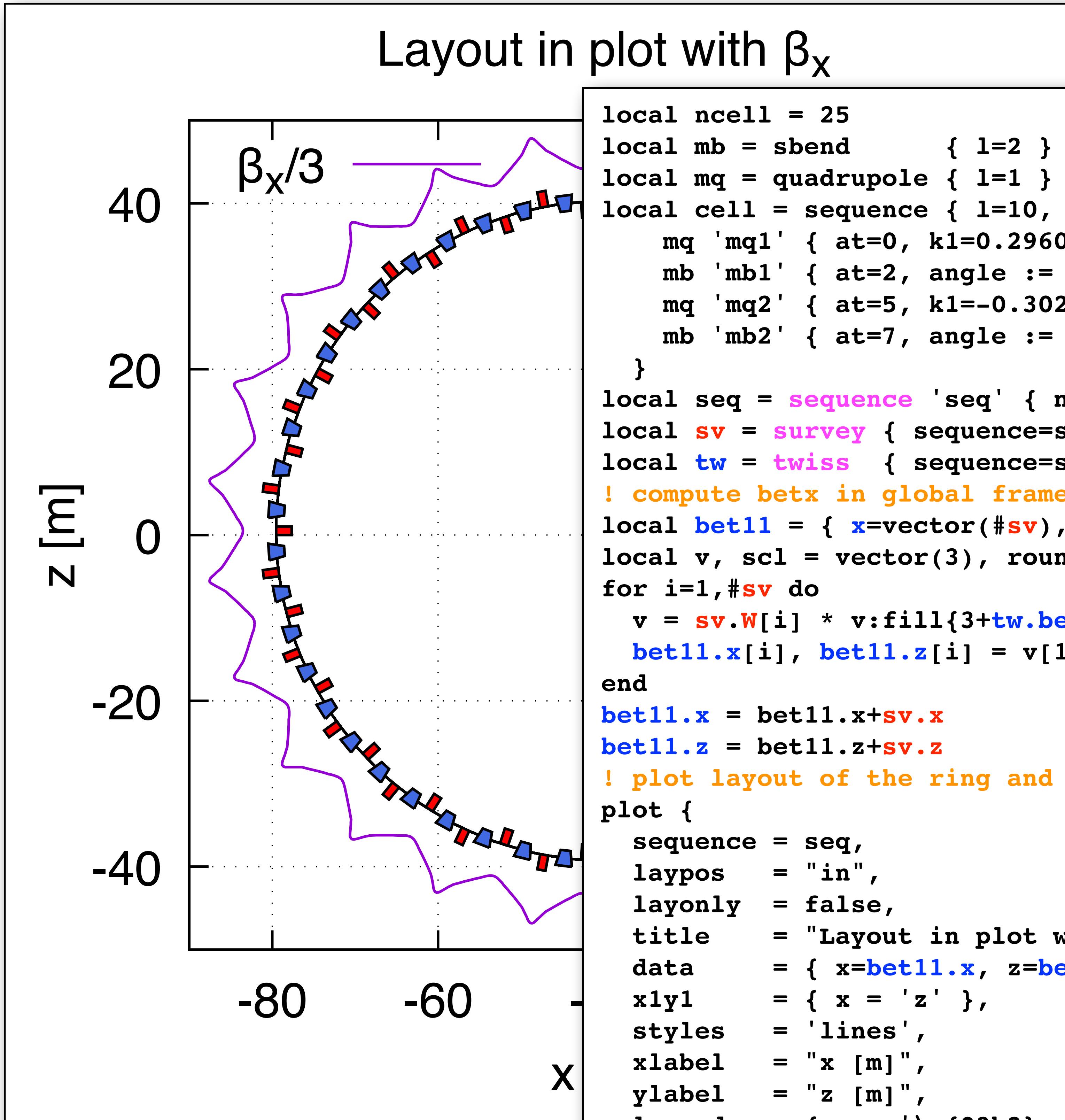


Plot survey & twiss *(two rings with β_x)*




```
local ncell = 25
local mb = sbend      { l=2 }
local mq = quadrupole { l=1 }
local cell = sequence { l=10, refer='entry',
    mq 'mq1' { at=0, k1=0.29601         },
    mb 'mb1' { at=2, angle := pi/ncell },
    mq 'mq2' { at=5, k1=-0.30242        },
    mb 'mb2' { at=7, angle := pi/ncell },
  }
local seq = sequence 'seq' { ncell*cell, beam=beam }
local sv = survey { sequence=seq, nslice=5, atslice=ftrue, mapsave=true }
local tw = twiss  { sequence=seq, nslice=5, atslice=ftrue }
! compute betx in global frame
local bet11 = { x=vector(#sv), z=vector(#sv) }
local v, scl = vector(3), round(tw.beta11:max()/5)
for i=1,#sv do
  v = sv.W[i] * v:fill{3+tw.beta11[i]/scl, 0, 0}
  bet11.x[i], bet11.z[i] = v[1], v[3]
end
bet11.x = bet11.x+sv.x
bet11.z = bet11.z+sv.z
! plot layout of the ring and the betx
plot {
  sequence = seq,
  laypos   = "in",
  layonly  = false,
  title    = "Layout in plot with \u{03b2}_x",
  data     = { x=bet11.x, z=bet11.z },
  x1y1     = { x = 'z' },
  styles   = 'lines',
  xlabel   = "x [m]",
  ylabel   = "z [m]",
  legend   = { z = '\u{03b2}_x/'..scl },
}
```

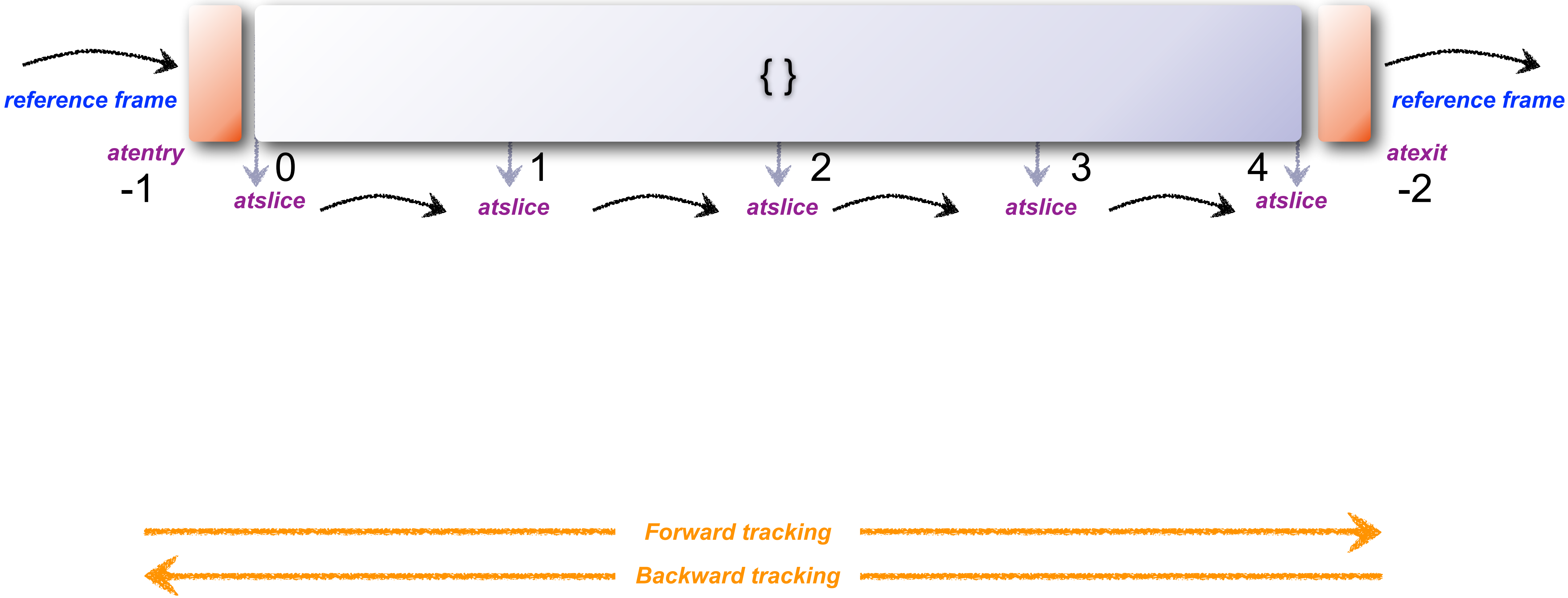
Plot survey & twiss *(two rings with β_x)*




```
local ncell = 25
local mb = sbend      { l=2 }
local mq = quadrupole { l=1 }
local cell = sequence { l=10, refer='entry',
    mq 'mq1' { at=0, k1=0.29601         },
    mb 'mb1' { at=2, angle := pi/ncell },
    mq 'mq2' { at=5, k1=-0.30242        },
    mb 'mb2' { at=7, angle := pi/ncell },
  }
local seq = sequence 'seq' { ncell*cell, beam=beam }
local sv = survey { sequence=seq, nslice=5, atslice=ftrue, mapsave=true }
local tw = twiss  { sequence=seq, nslice=5, atslice=ftrue }
! compute betx in global frame
local bet11 = { x=vector(#sv), z=vector(#sv) }
local v, scl = vector(3), round(tw.beta11:max()/5)
for i=1,#sv do
  v = sv.W[i] * v:fill{3+tw.beta11[i]/scl, 0, 0}
  bet11.x[i], bet11.z[i] = v[1], v[3]
end
bet11.x = bet11.x+sv.x
bet11.z = bet11.z+sv.z
! plot layout of the ring and the betx
plot {
  sequence = seq,
  laypos   = "in",
  layonly  = false,
  title    = "Layout in plot with \u{03b2}_x",
  data     = { x=bet11.x, z=bet11.z },
  x1y1     = { x = 'z' },
  styles   = 'lines',
  xlabel   = "x [m]",
  ylabel   = "z [m]",
  legend   = { z = '\u{03b2}_x/'..scl },
}
```

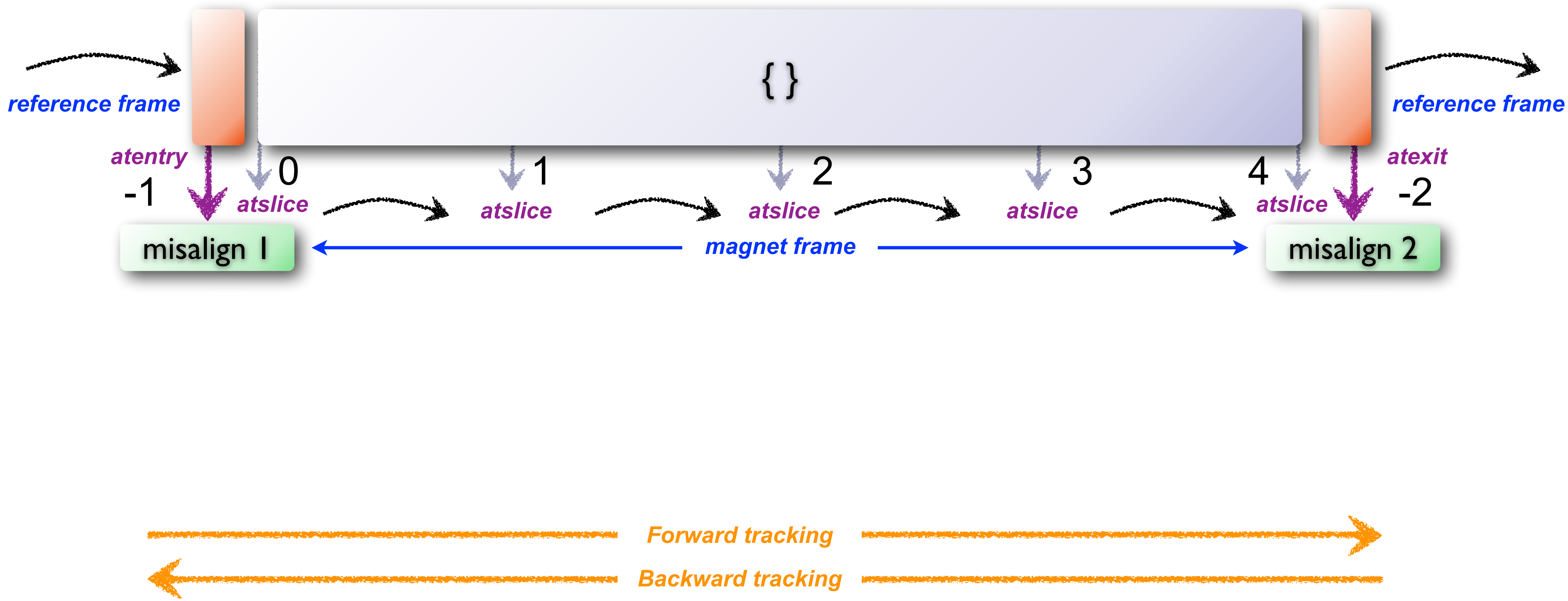
Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Element tracking: slices, actions & frames

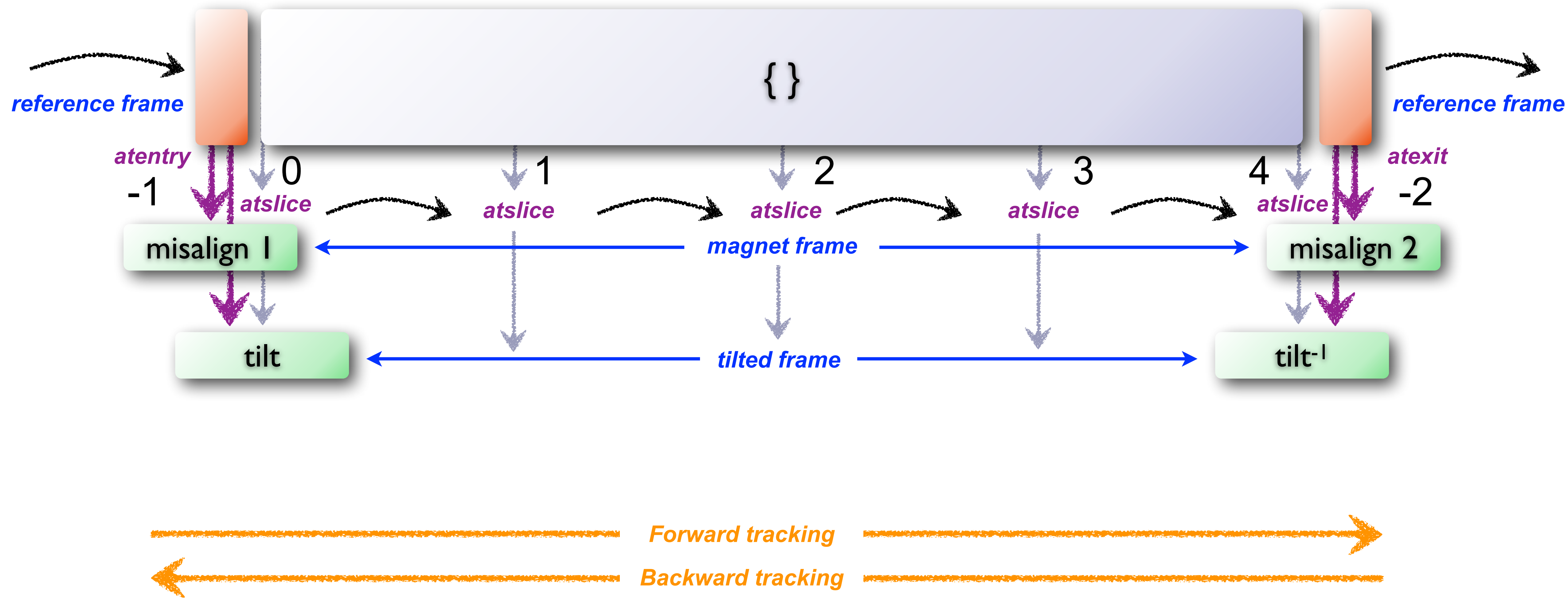


Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Element tracking: slices, actions & frames

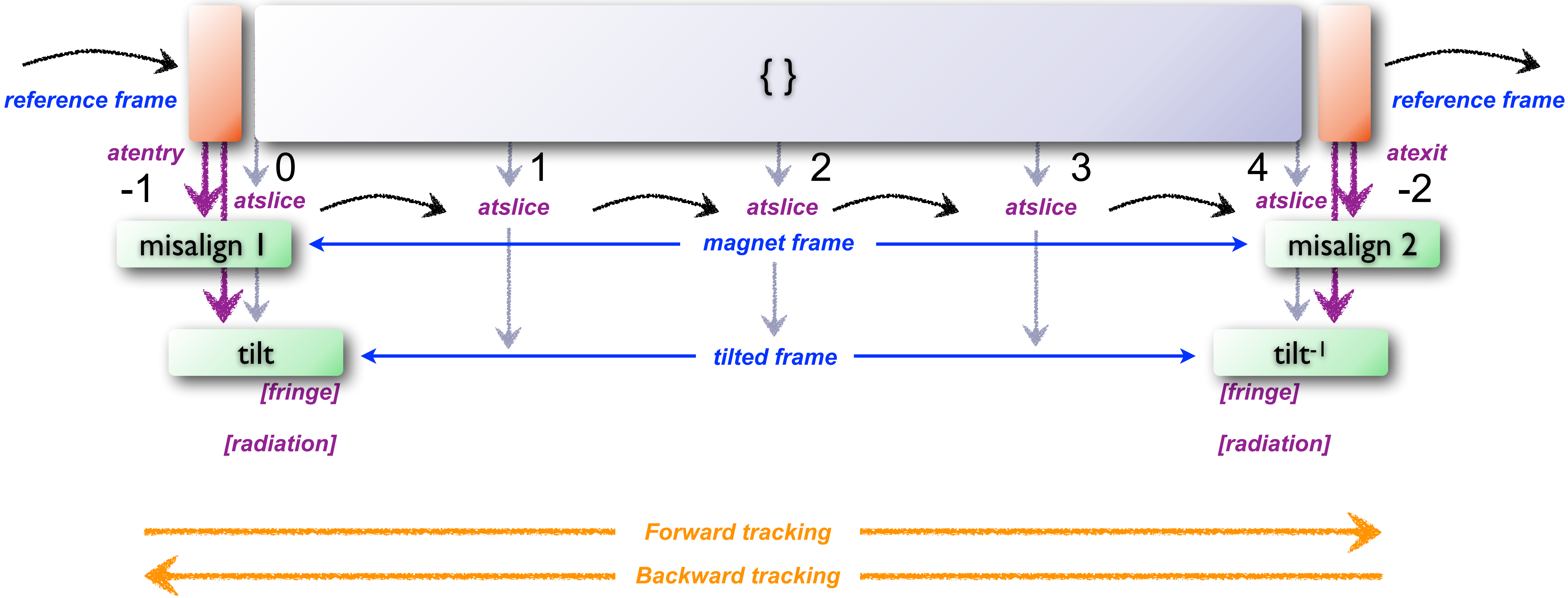


Element tracking: slices, actions & frames



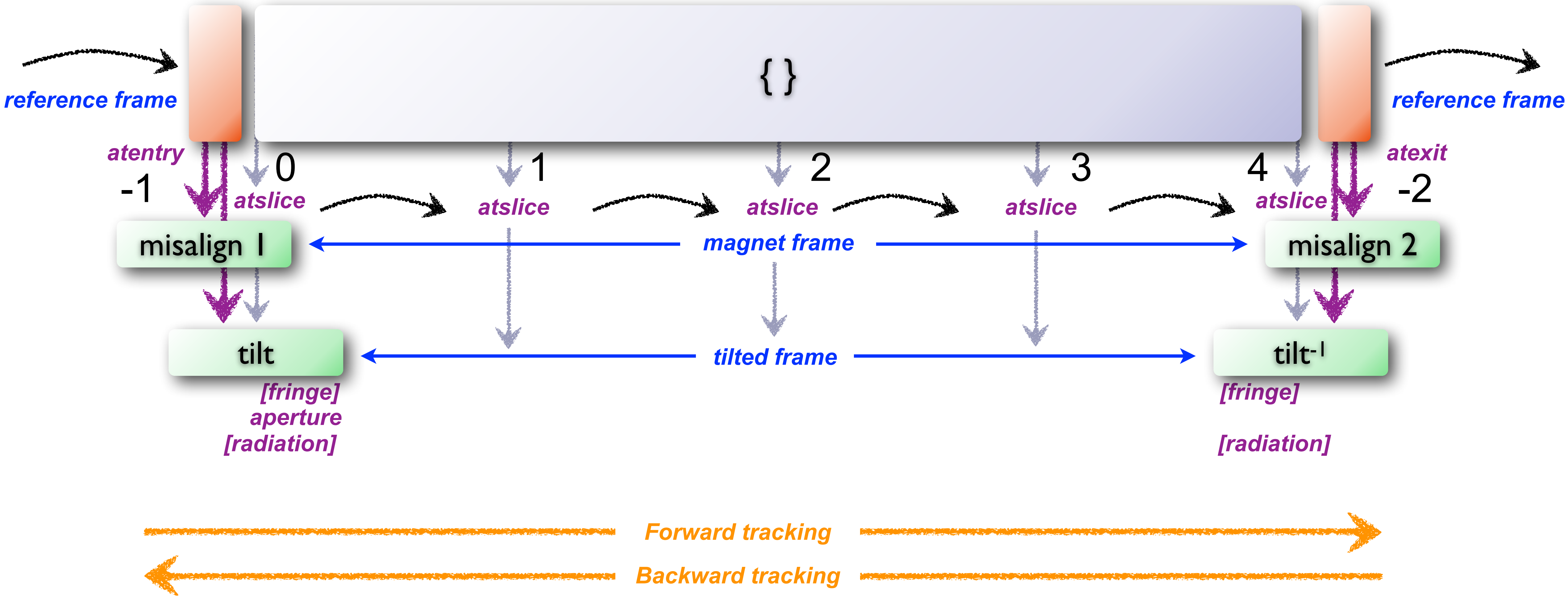
Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Element tracking: slices, actions & frames

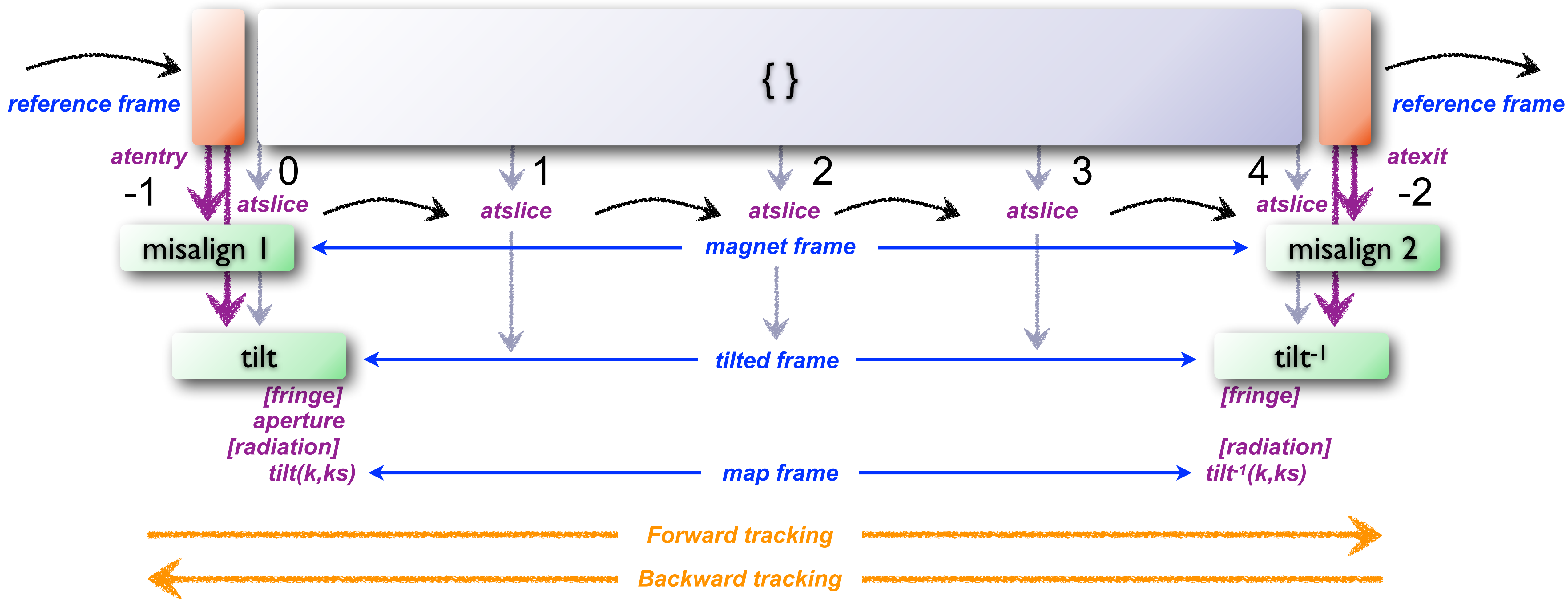


Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

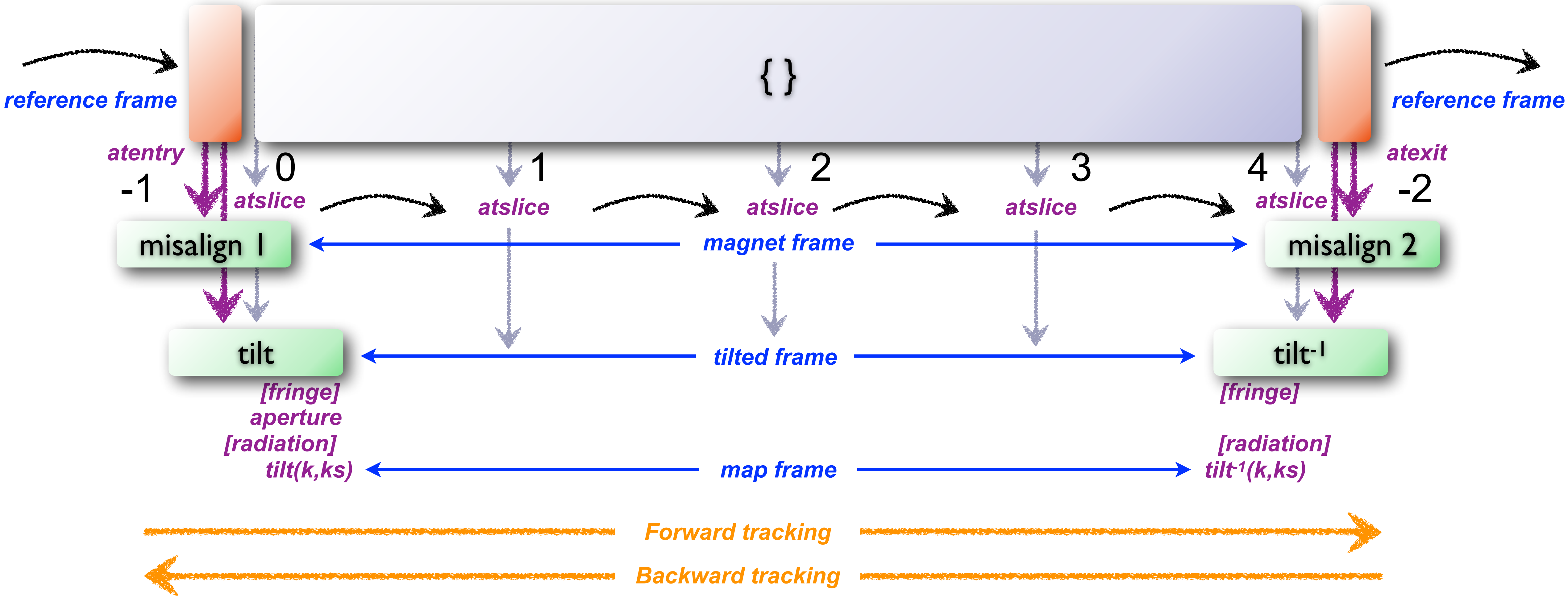
Element tracking: slices, actions & frames



Element tracking: slices, actions & frames



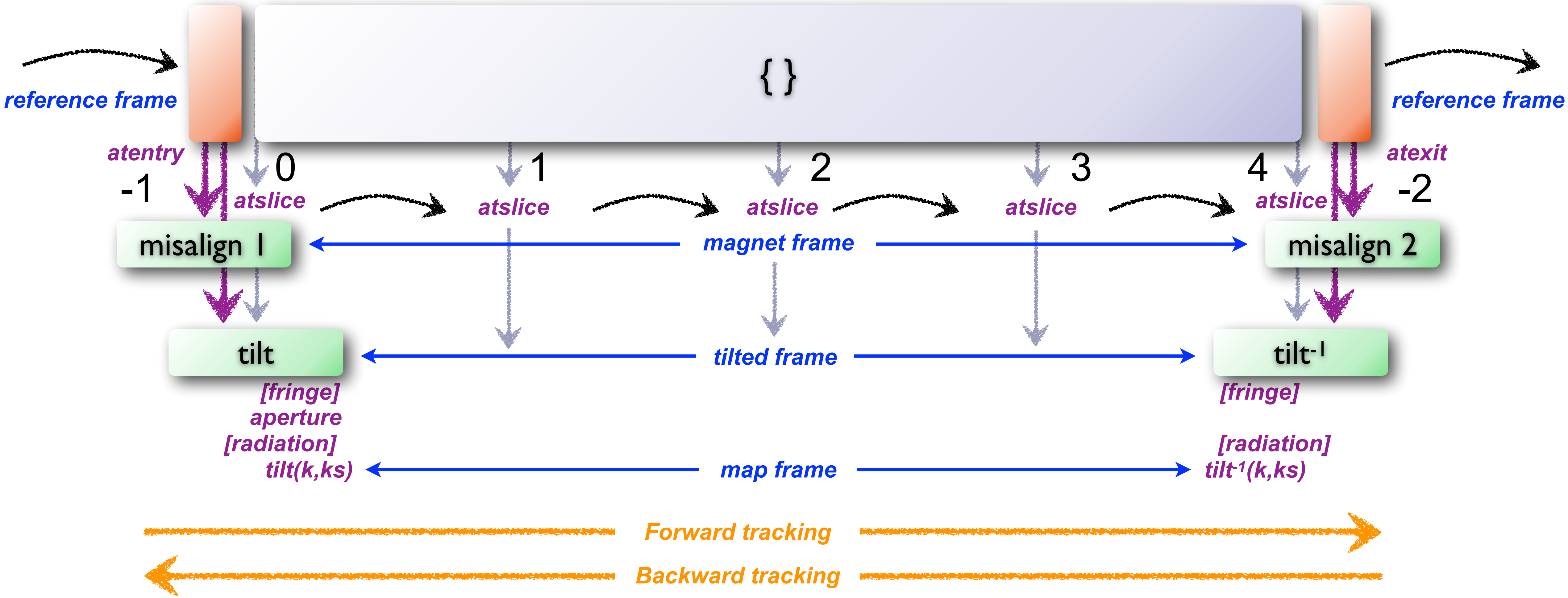
Element tracking: slices, actions & frames




```
atentry(elm, m,  sdir, -1)
mis    (elm, m,  sdir)
rot    (tlt, m,  sdir)
fringe (elm, m,  sdir)
track  (elm, m,    1 , thick, thin)
fringe (elm, m, -sdir)
rot    (tlt, m, -sdir)
mis    (elm, m, -sdir)
atexit (elm, m, -sdir, -2)
```

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Element tracking: slices, actions & frames



- **Slicing** can be uniform or arbitrary.
- **Subelements** thick or thin can be inserted at arbitrary relative (to parent length) or absolute (from parent entry) positions. Subelements define slices.
- **Installing** elements in sequence automatically (user-policy) insert them as subelement upon collision.

```
atentry(elm, m,  sdir, -1)
mis    (elm, m,  sdir)
rot    (tlt, m,  sdir)
fringe (elm, m,  sdir)
track  (elm, m,    1 , thick, thin)
fringe (elm, m, -sdir)
rot    (tlt, m, -sdir)
mis    (elm, m, -sdir)
atexit (elm, m, -sdir, -2)
```

- **Misalignments** (element to sequence) restore the frame on exit. Permanent misalignments (element property) don't (i.e. **patches**). Survey can consider misalignments (user-policy) for superposition inside elements.

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

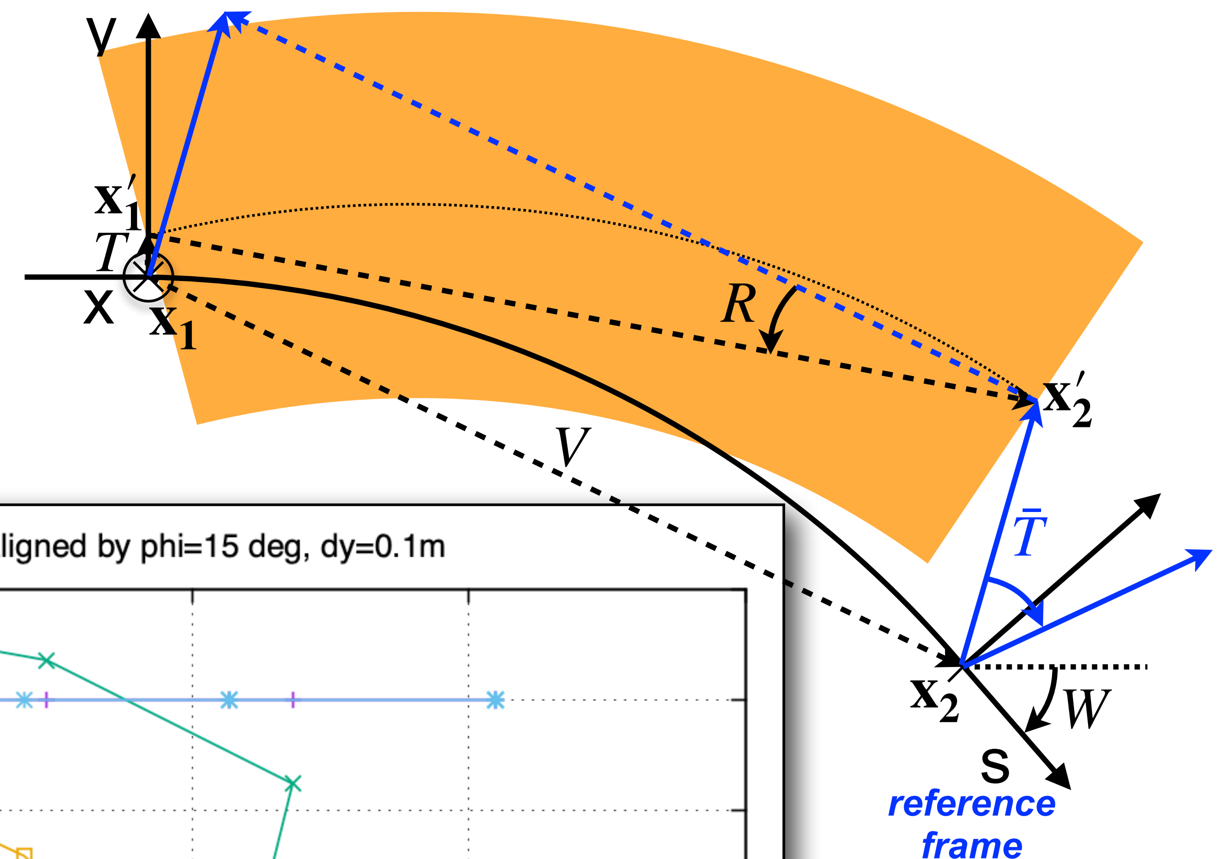
Survey: sbend tilted by 90° — dphi 15° dy 0.1m

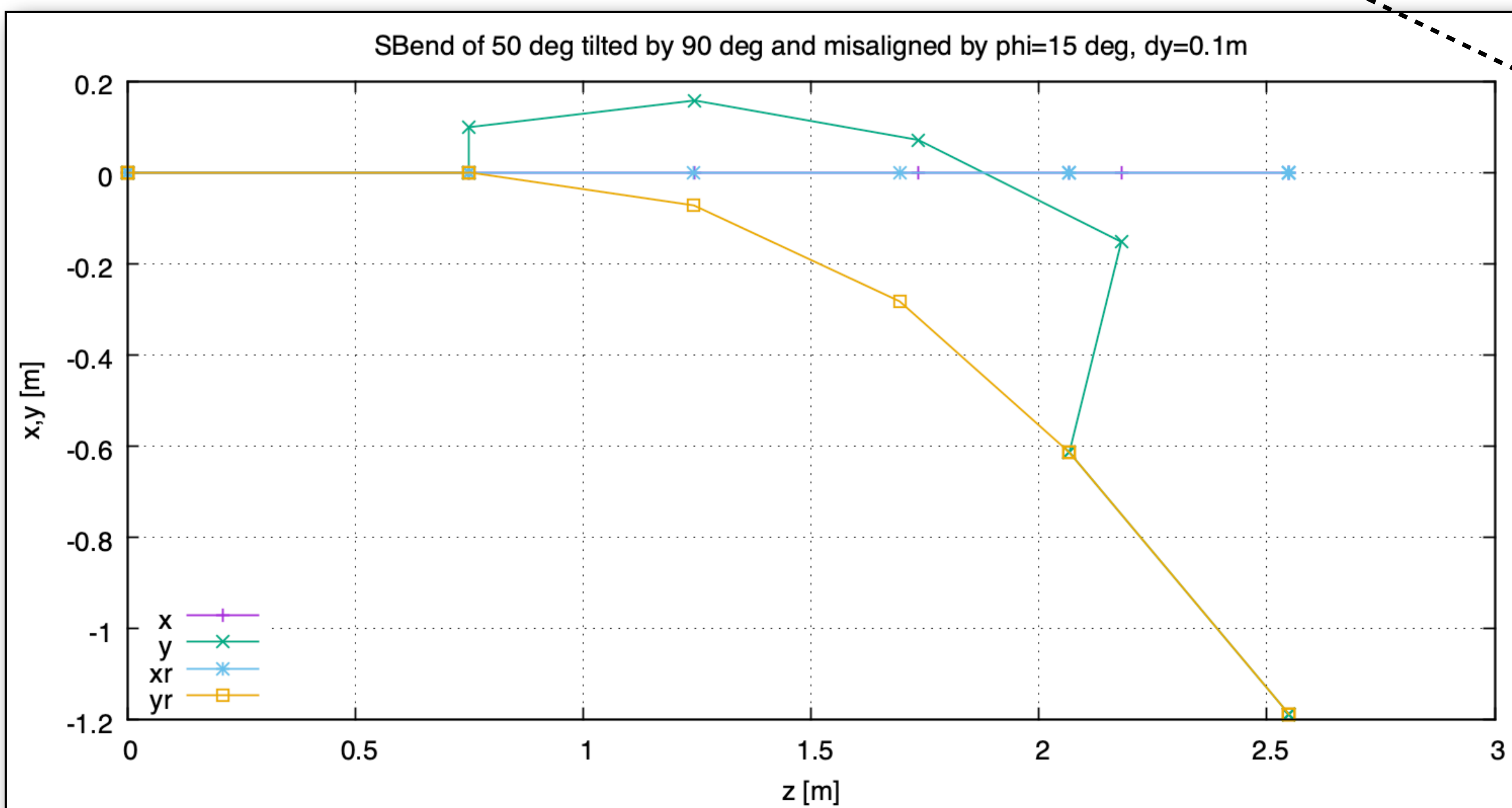
Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Survey: sbend tilted by 90° — dphi 15° dy 0.1m

Beams Department

x, y with misalignments, xr, yr reference frame without misalignment





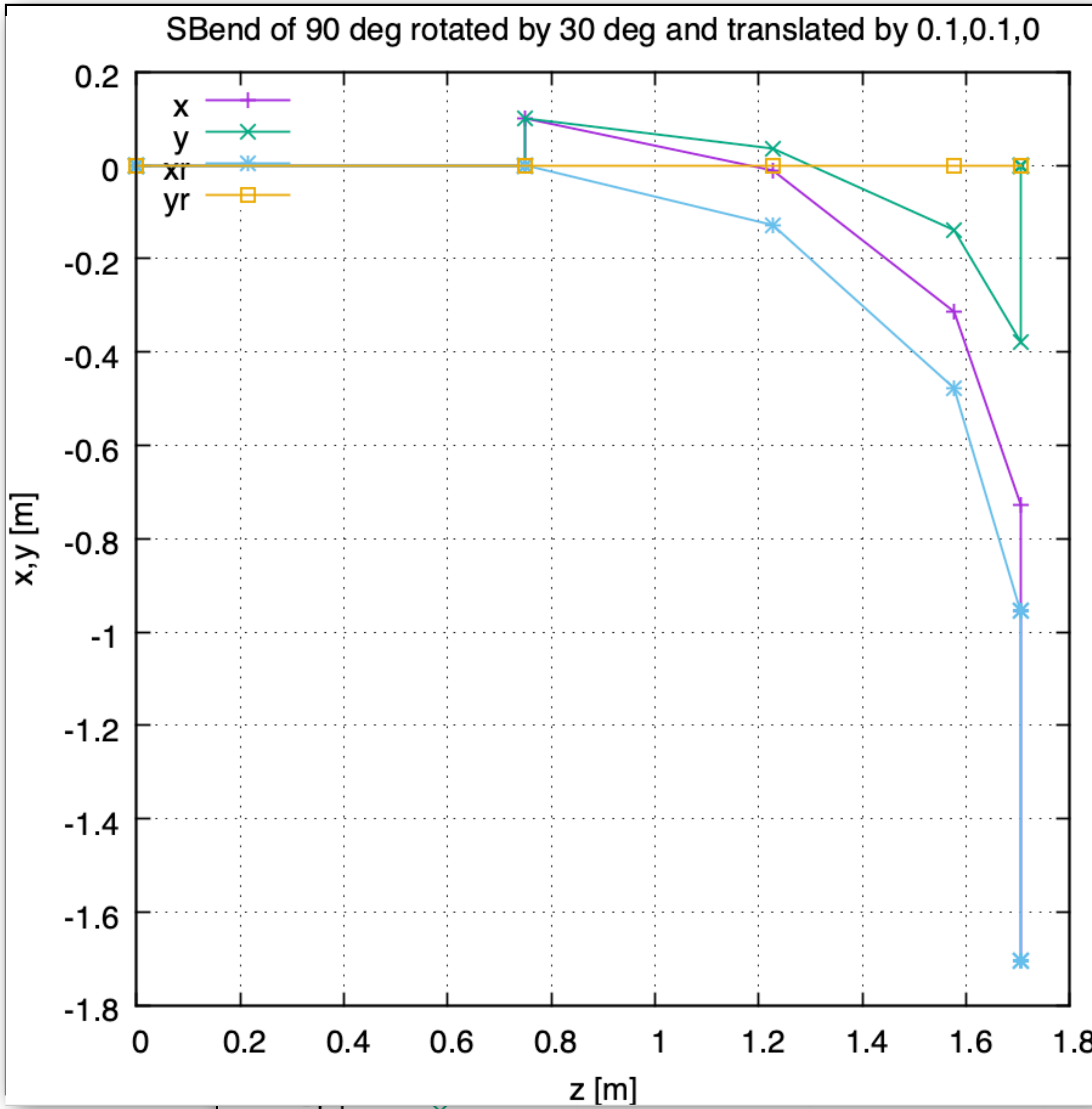
nslice = 3

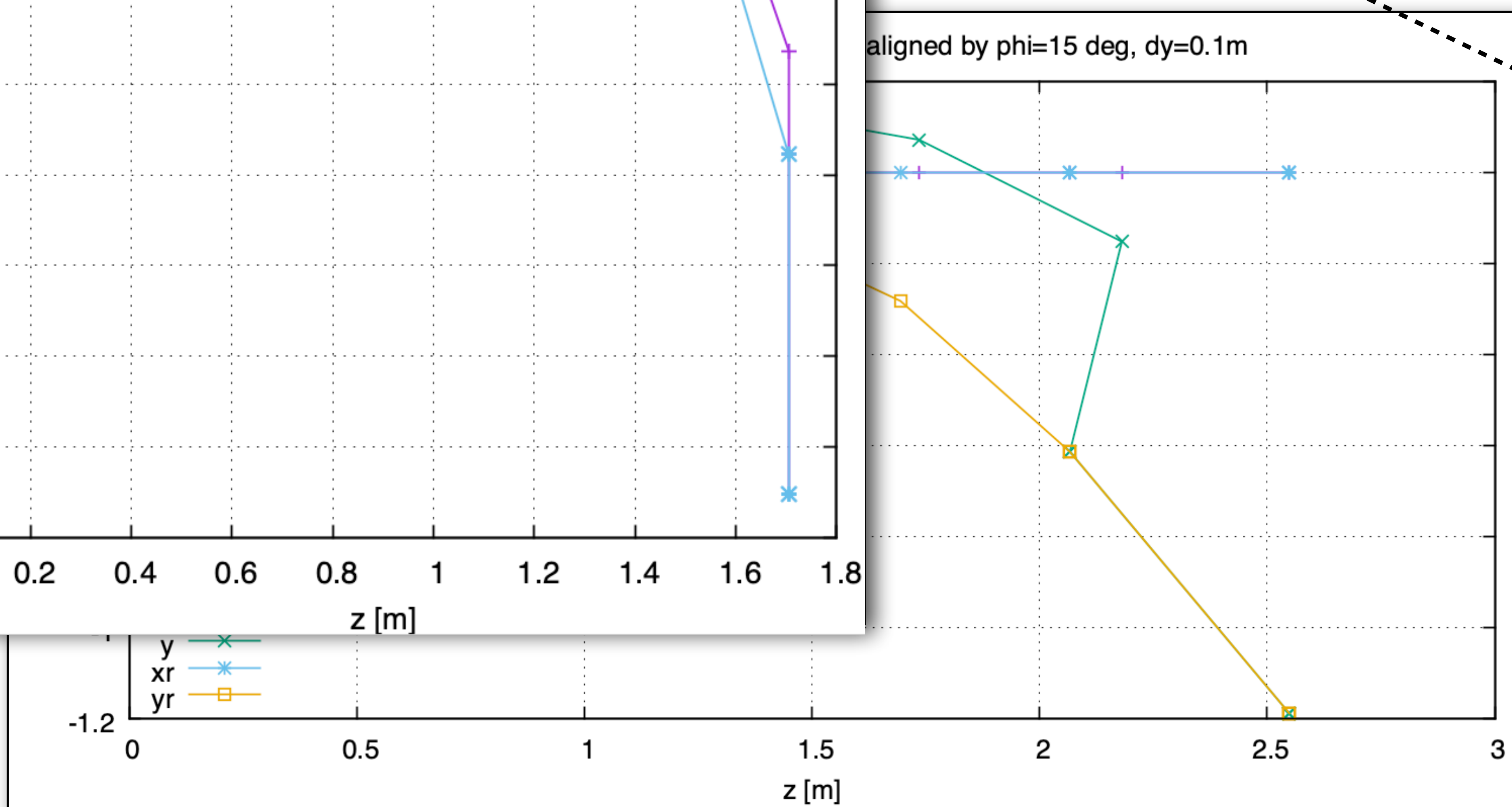
Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

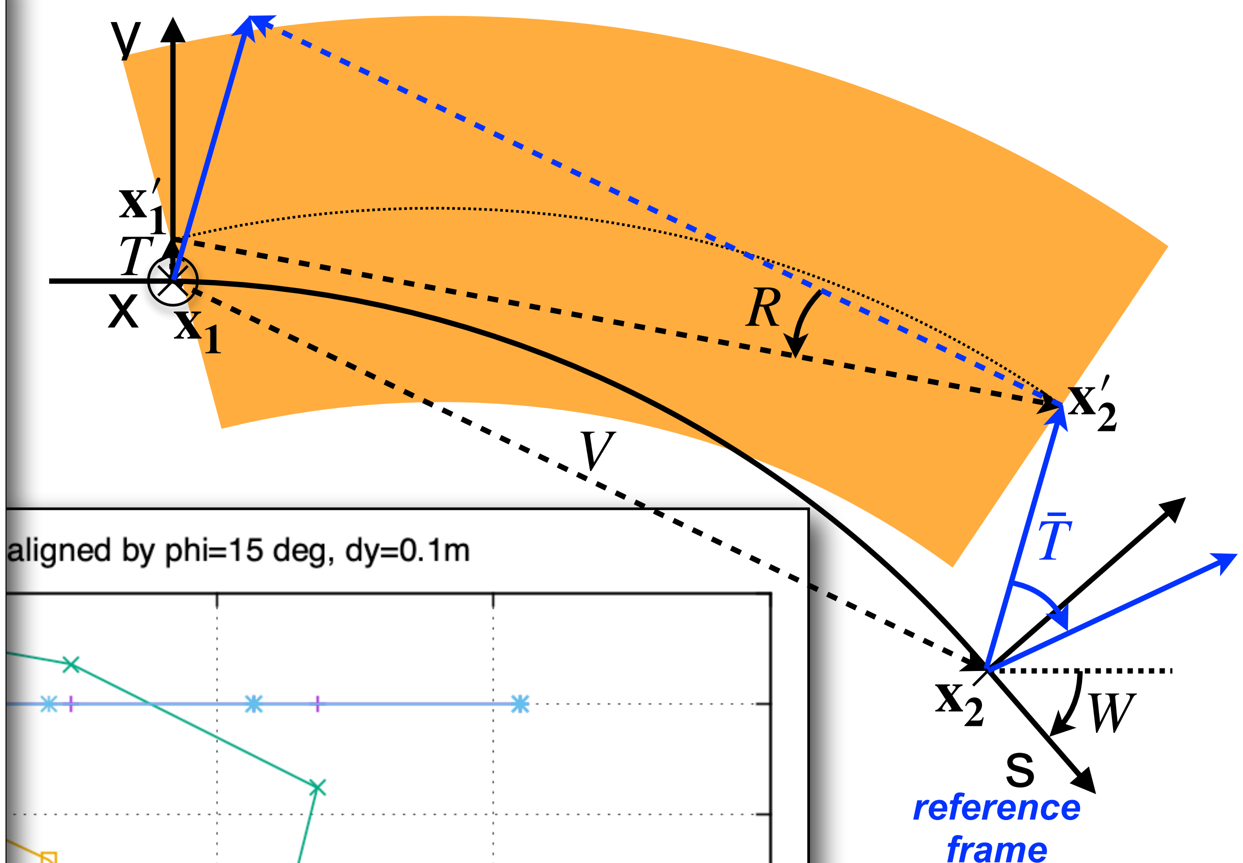
Survey: sbend tilted by 90° — dphi 15° dy 0.1m

BE Beams Department

x, y with misalignments, xr, yr reference frame without misalignment







nslice = 3

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Tracking actions *(Survey, Track, Cofind and Twiss)*

Tracking actions *(Survey, Track, Cofind and Twiss)*

- **Actions are functions** (or objects with function-like semantic).
  - ➡ MAD-NG functions are *first class lexical closures* (fun & env) and can do everything…
    - i.e. high order functions that can receive **and** return multiple arguments.
  - ➡ actions kinds: `atentry`, `atslice`, `atexit`, *`atapen`*, *`atsave`*.
  - ➡ **mechanism to customise or extend other commands** (e.g. **Twiss** with **Track** and **Cofind**).

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Tracking actions *(Survey, Track, Cofind and Twiss)*

- **Actions are functions** (or objects with function-like semantic).
  - ➡ MAD-NG functions are *first class lexical closures* (fun & env) and can do everything…
    - i.e. high order functions that can receive **and** return multiple arguments.
  - ➡ actions kinds: `atentry`, `atslice`, `atexit`, *`ataper`*, *`atsave`*.
  - ➡ **mechanism to customise or extend other commands** (e.g. **Twiss** with **Track** and **Cofind**).
- Actions can be **combined** with combinators (and selectors).
  - ➡ `chain(f₁,f₂)` ➠ `f₁() ; return f₂()`.
  - ➡ `achain(f₁,f₂)` ➠ `return f₁() and f₂()`.
  - ➡ `ochain(f₁,f₂)` ➠ `return f₁() or f₂()`.
  - ➡ `compose(f₁,f₂)` ➠ `return f₁(f₂())`.
  - ➡ `ftrue, ffalse, fnone`.

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Tracking actions *(Survey, Track, Cofind and Twiss)*

- **Actions are functions** (or objects with function-like semantic).
  - ➡ MAD-NG functions are *first class lexical closures* (fun & env) and can do everything…
    - i.e. high order functions that can receive **and** return multiple arguments.
  - ➡ actions kinds: `atentry`, `atslice`, `atexit`, *`ataper`*, *`atsave`*.
  - ➡ **mechanism to customise or extend other commands** (e.g. **Twiss** with **Track** and **Cofind**).
- Actions can be **combined** with combinators (and selectors).

```
➡ chain(f1,f2)   ➠ f1() ; return f2().
➡ achain(f1,f2)  ➠ return f1() and f2().
➡ ochain(f1,f2)  ➠ return f1() or f2().
➡ compose(f1,f2) ➠ return f1(f2()).
➡ ftrue, ffalse, fnone.
```

- Actions can be **selected** by *selectors*:
  - ➡ Selectors are functions to enable/disable actions based on some particular criteria e.g. slices number or any other user-defined criteria.
    predefined selectors: atall, atentry, atbegin, atbody, atbound, atend, atexit, atmid, atins, atstd, actionat, action.

Tracking actions *(Survey, Track, Cofind and Twiss)*

- **Actions are functions** (or objects with function-like semantic).
  - MAD-NG functions are *first class lexical closures* (fun & env) and can do everything…
    - i.e. high order functions that can receive **and** return multiple arguments.
  - actions kinds: `atentry`, `atslice`, `atexit`, *`ataper`*, *`atsave`*.
  - **mechanism to customise or extend other commands** (e.g. **Twiss** with **Track** and **Cofind**).
- Actions can be **combined** with combinators (and selectors).
  - `chain(f₁,f₂)` ➠ `f₁() ; return f₂().`
  - `achain(f₁,f₂)` ➠ `return f₁() and f₂().`
  - `ochain(f₁,f₂)` ➠ `return f₁() or f₂().`
  - `compose(f₁,f₂)` ➠ `return f₁(f₂()).`
  - `ftrue, ffalse, fnone.`
- Actions can be **selected** by *selectors*:
  - Selectors are functions to enable/disable actions based on some particular criteria e.g. slices number or any other user-defined criteria.
    predefined selectors: atall, atentry, atbegin, atbody, atbound, atend, atexit, atmid, atins, atstd, actionat, action.
- **Actions are triggered by tracking codes** (**Survey** and **Track**).
  - actions are `chained` so they are independent from each other.
  - default for *ataper*: check for aperture *at slice* 0 (**titled frame**).
  - default for *atsave*: save data *at exit* (**reference frame**), and *at slices* (titled frame) if atslice = ftrue.

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Tracking actions *(Survey, Track, Cofind and Twiss)*

- **Actions are functions** (or objects with function-like semantic).
  - ➡ MAD-NG functions are *first class lexical closures* (fun & env) and can do everything…
    - i.e. high order functions that can receive **and** return multiple arguments.
  - ➡ actions kinds: `atentry`, `atslice`, `atexit`, *`ataper`*, *`atsave`*.
  - ➡ **mechanism to customise or extend other commands** (e.g. **Twiss** with **Track** and **Cofind**).
- Actions can be **combined** with combinators (and selectors).
  - ➡ chain(f_1,f_2) ➠ f_1() ; return f_2().
  - ➡ achain(f_1,f_2) ➠ return f_1() **and** f_2().
  - ➡ ochain(f_1,f_2) ➠ return f_1() **or** f_2().
  - ➡ compose(f_1,f_2) ➠ return f_1(f_2()).
  - ➡ ftrue, ffalse, fnone.
- Actions can be **selected** by *selectors*:
  - ➡ Selectors are functions to enable/disable actions … particular criteria e.g. slices number or any other user-defined crit…
    predefined selectors: atall, atentry, atbegin, atbody, atbound, atend, atexit, atmid, atins, atstd, actionat, action.

Actions are a powerful tool to extend tracking codes (survey and track). E.g. connect sequences (or beams) together; replace, extend or wrap computations; add extra physics between multi-particules or damaps, etc…

- **Actions are triggered by tracking codes** (**Survey** and **Track**).
  - ➡ actions are `chained` so they are independent from each other.
  - ➡ default for *ataper*: check for aperture *at slice* 0 (**titled frame**).
  - ➡ default for *atsave*: save data *at exit* (**reference frame**), and *at slices* (titled frame) if atslice = ftrue.

Order of execution at each slice

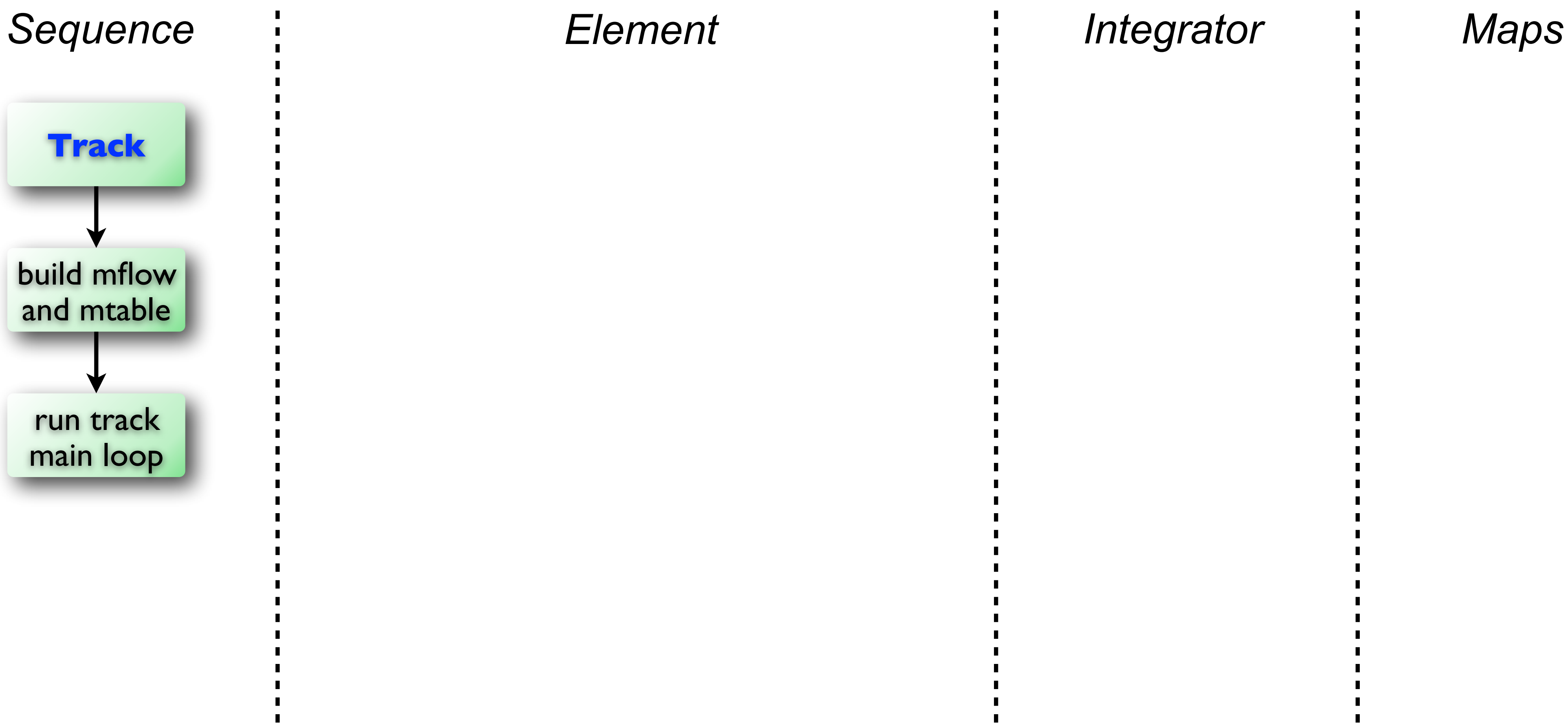
atslice = *ftrue*
atbegin and ataper
and ataper (user)
atsave (track)
and atsave (twiss)
and atsave (user)

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Track in “depth” : user-defined possible extensions

Sequence	*Element*	*Integrator*	*Maps*

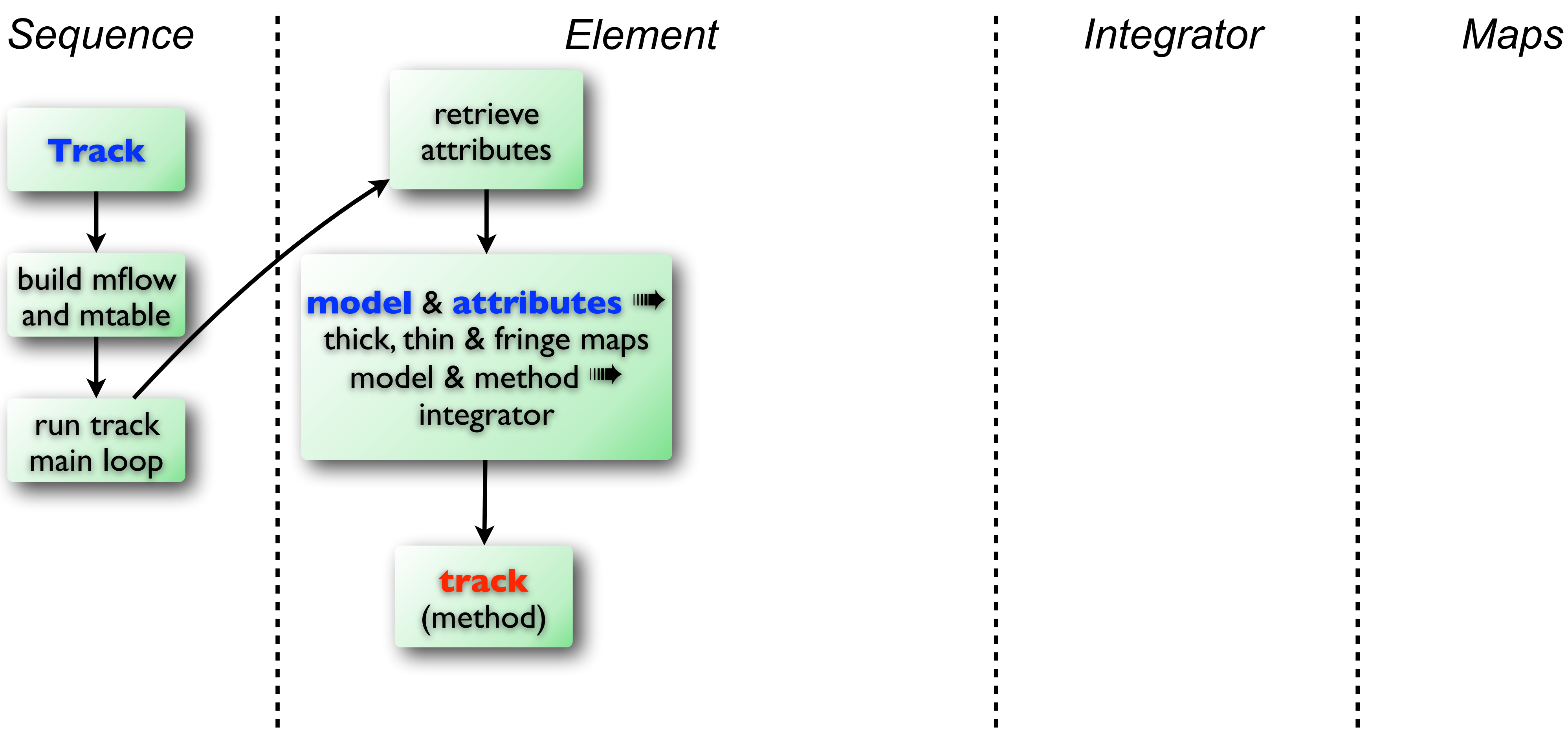
Track in “depth” : user-defined possible extensions



Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

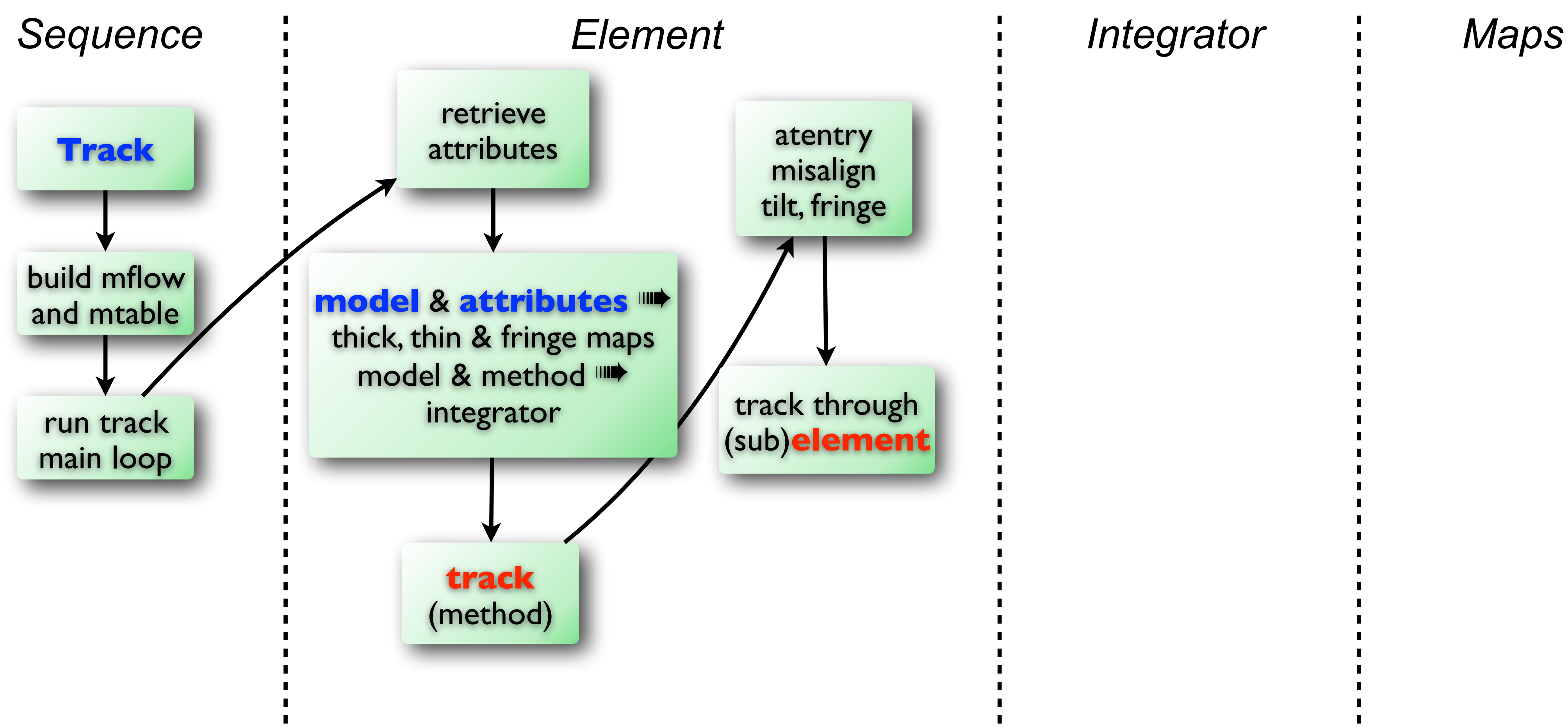
Track in "depth" : user-defined possible extensions

Beams Department



Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

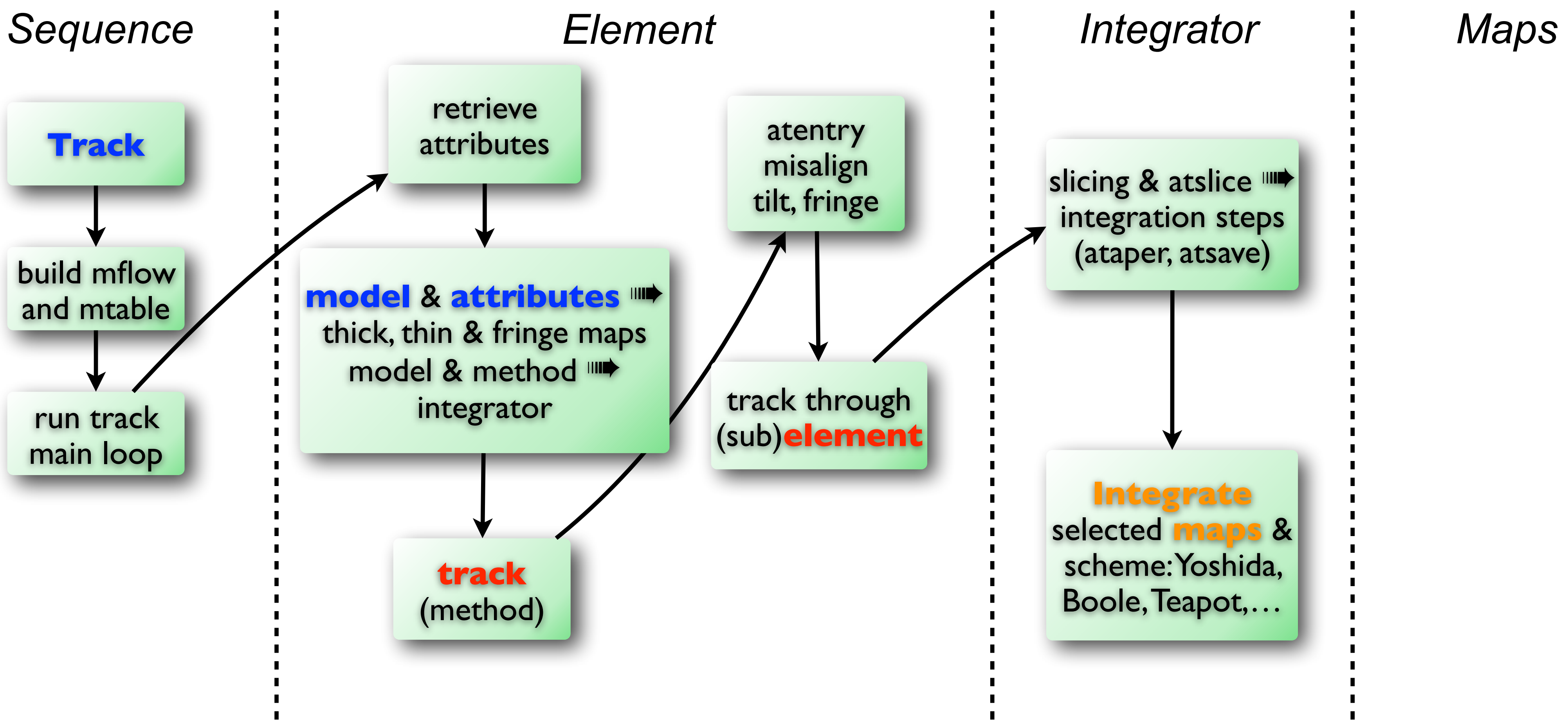
Track in "depth" : user-defined possible extensions



Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

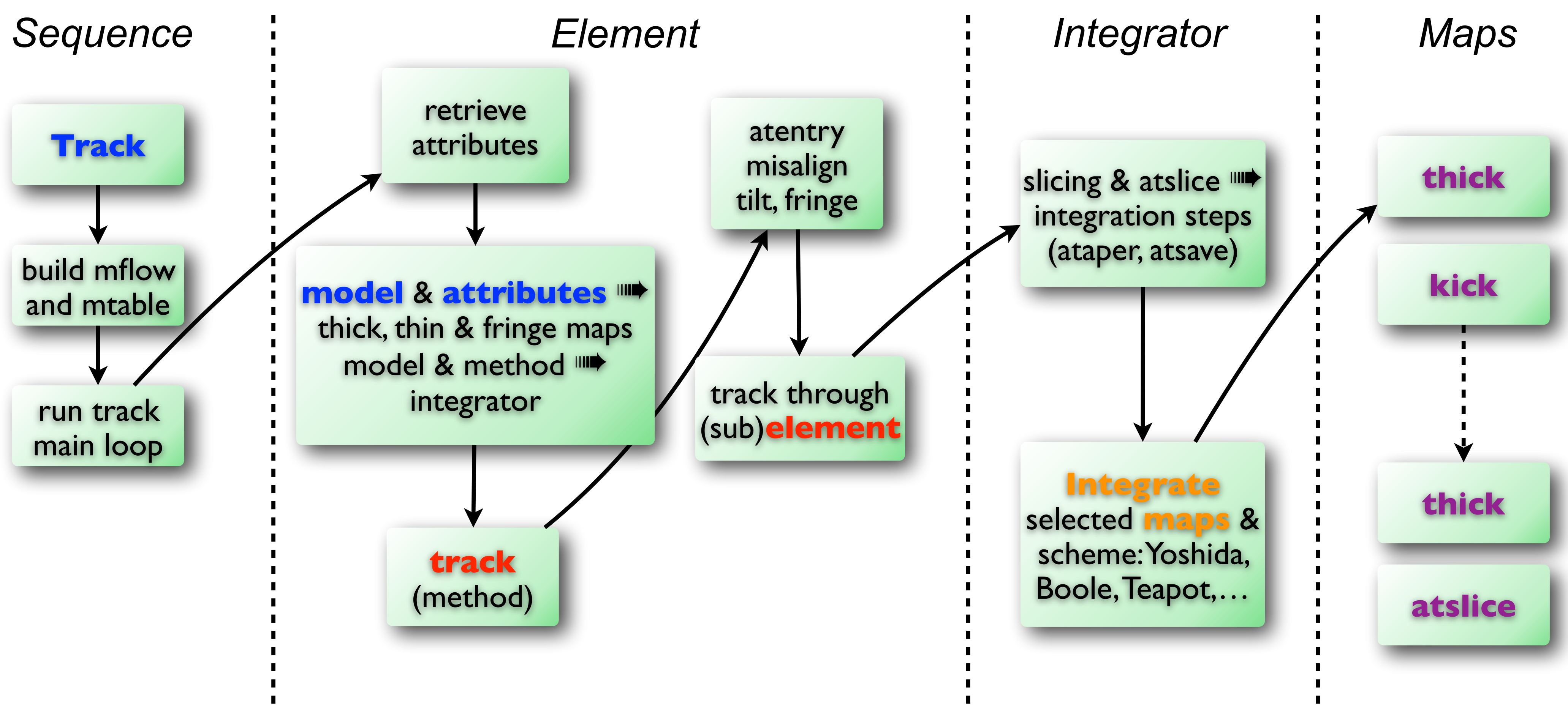
Track in “depth” : user-defined possible extensions

BE Beams Department



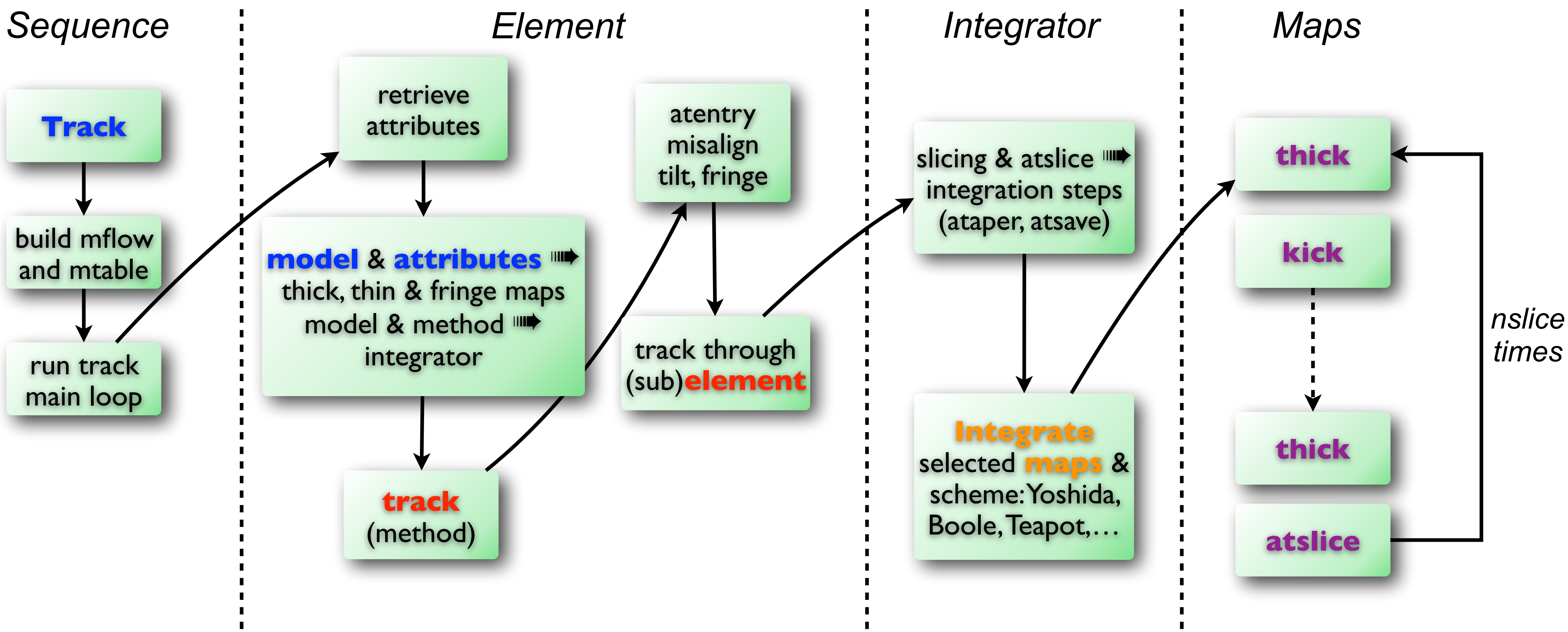
Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Track in “depth” : user-defined possible extensions



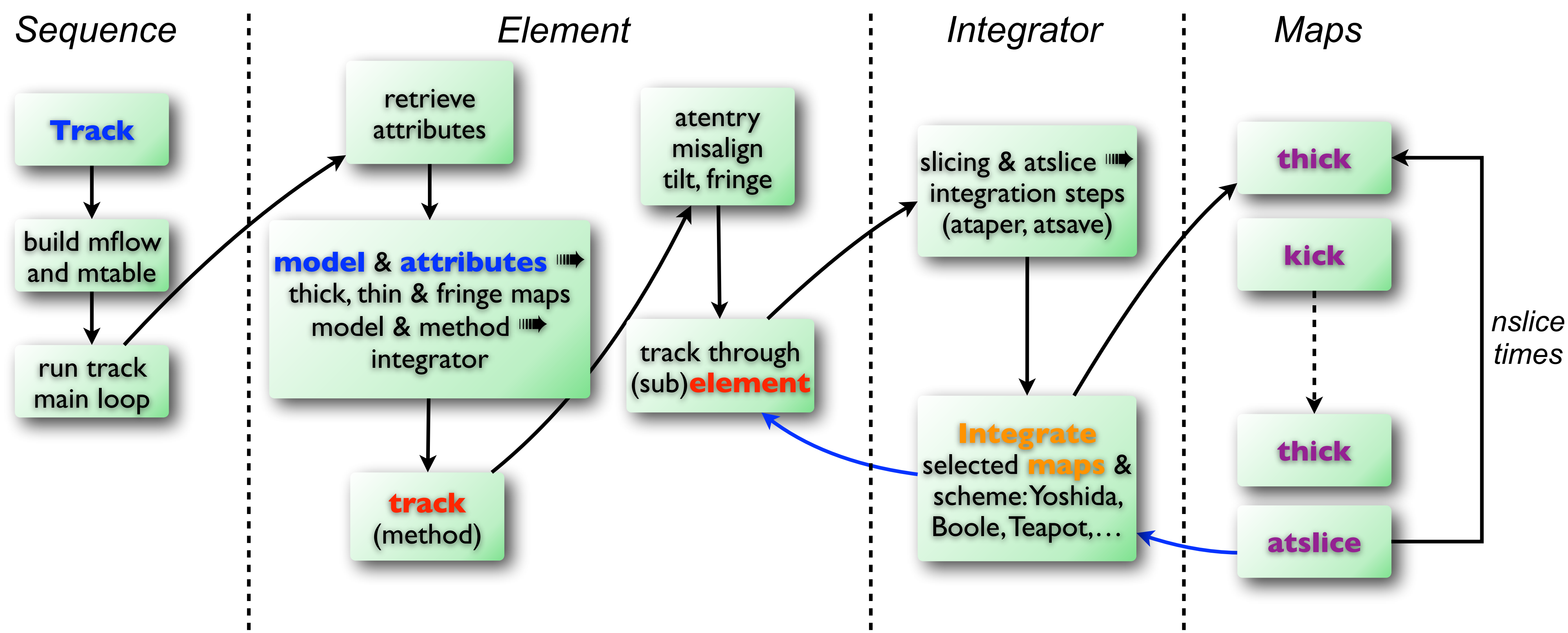
Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Track in “depth” : user-defined possible extensions



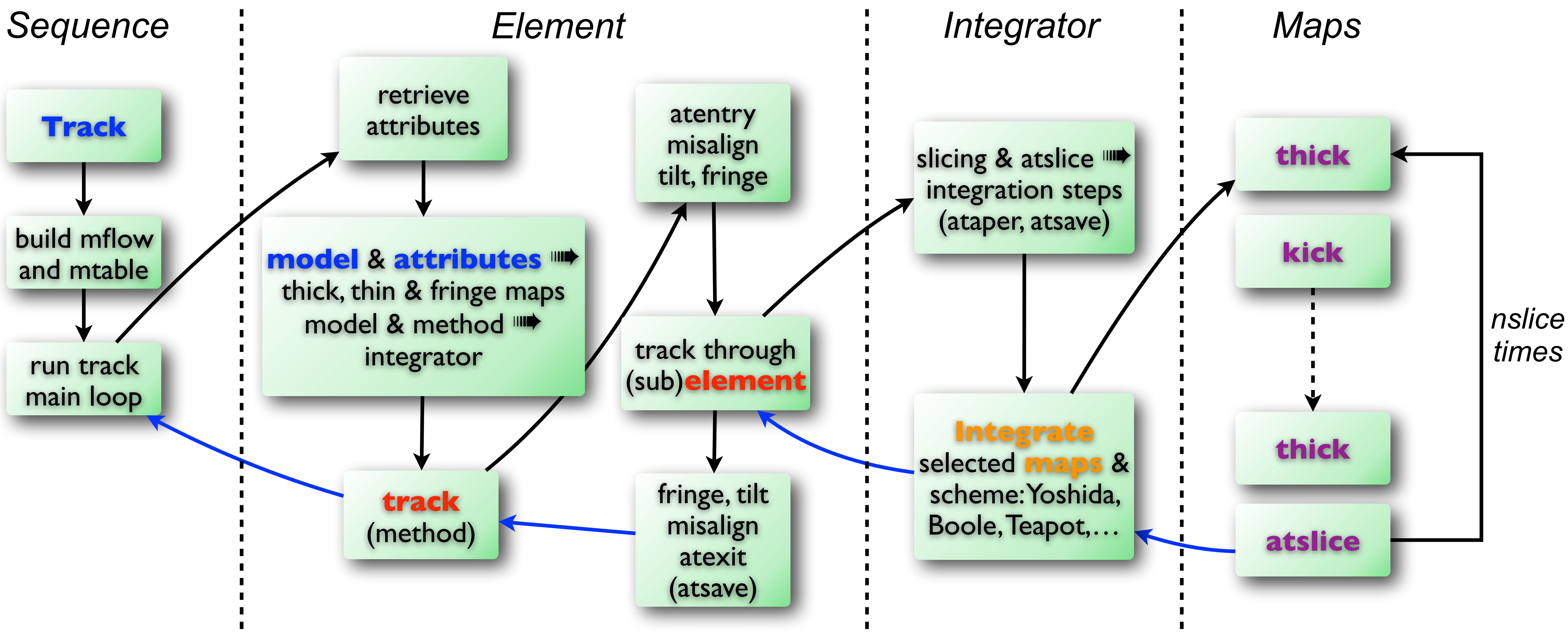
Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Track in “depth” : user-defined possible extensions



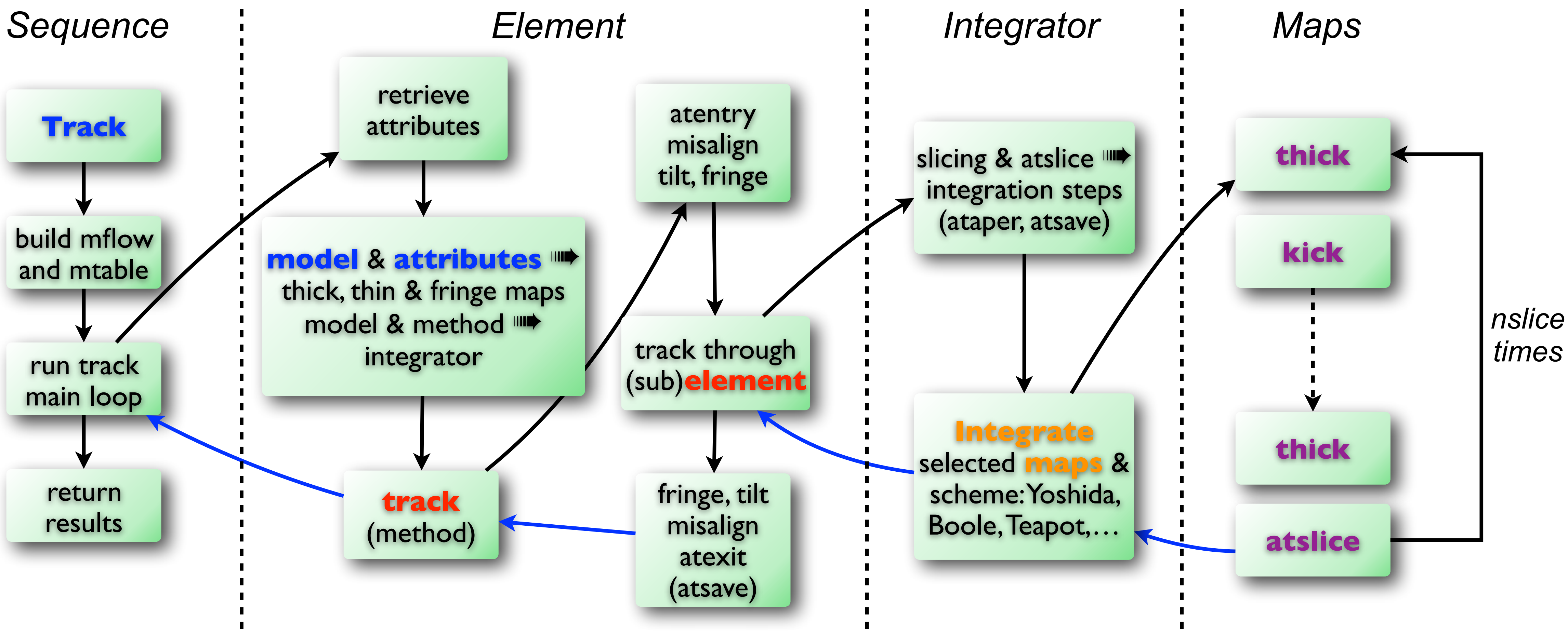
Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Track in “depth” : user-defined possible extensions



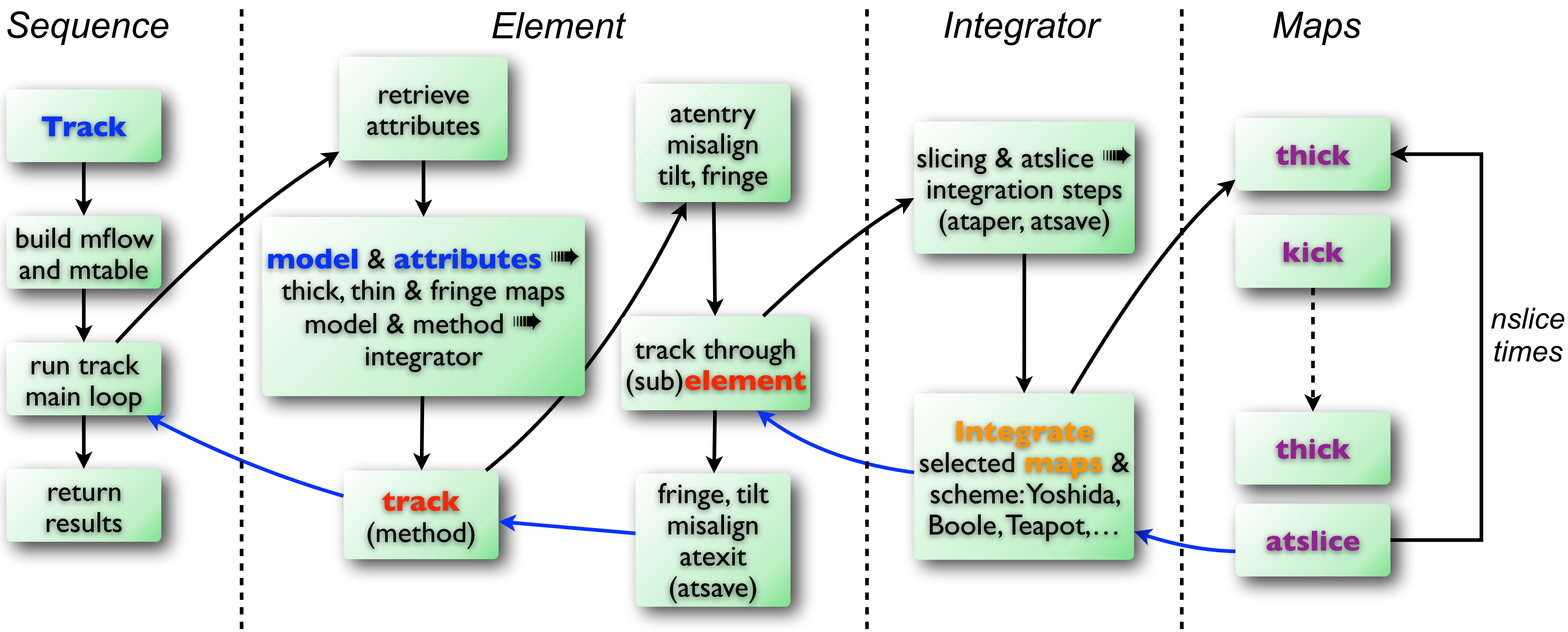
Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Track in "depth" : user-defined possible extensions



Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

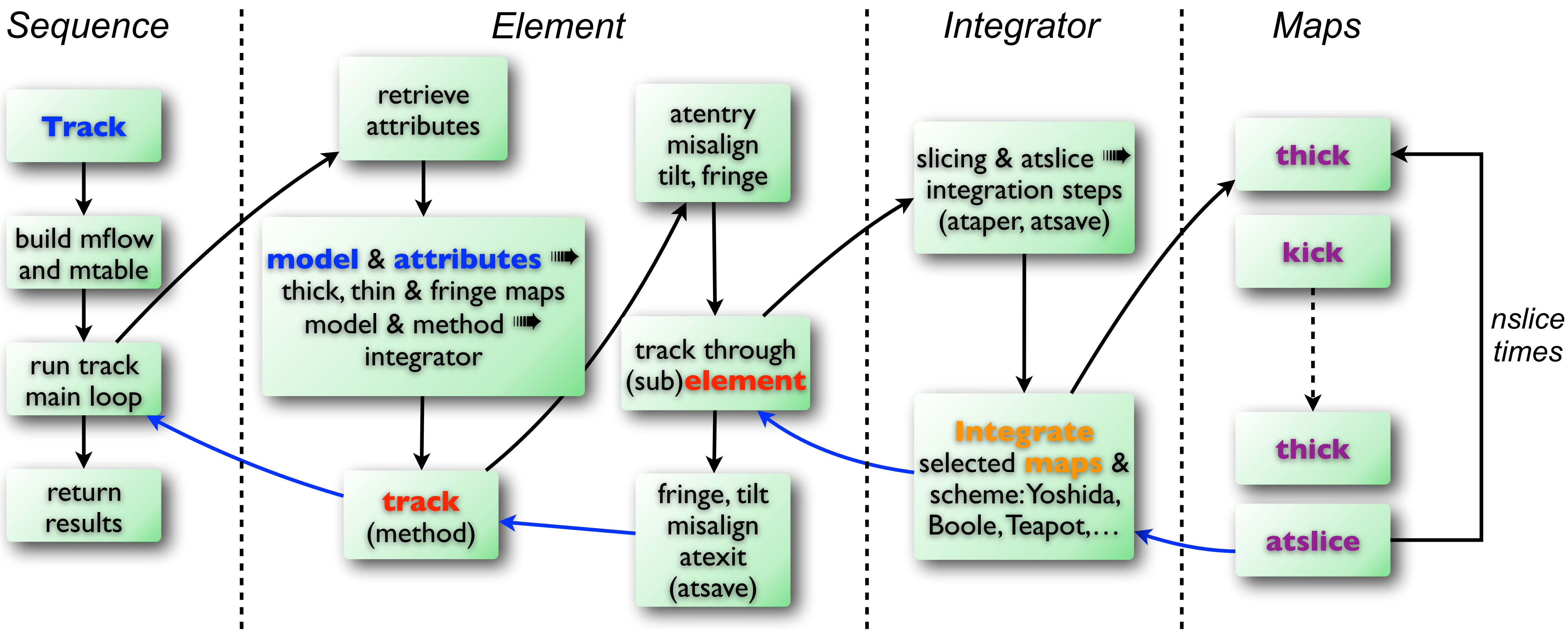
Track in "depth" : user-defined possible extensions



Physics can be parametrised and/or configured by element attributes and commands attributes

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Track in “depth” : user-defined possible extensions

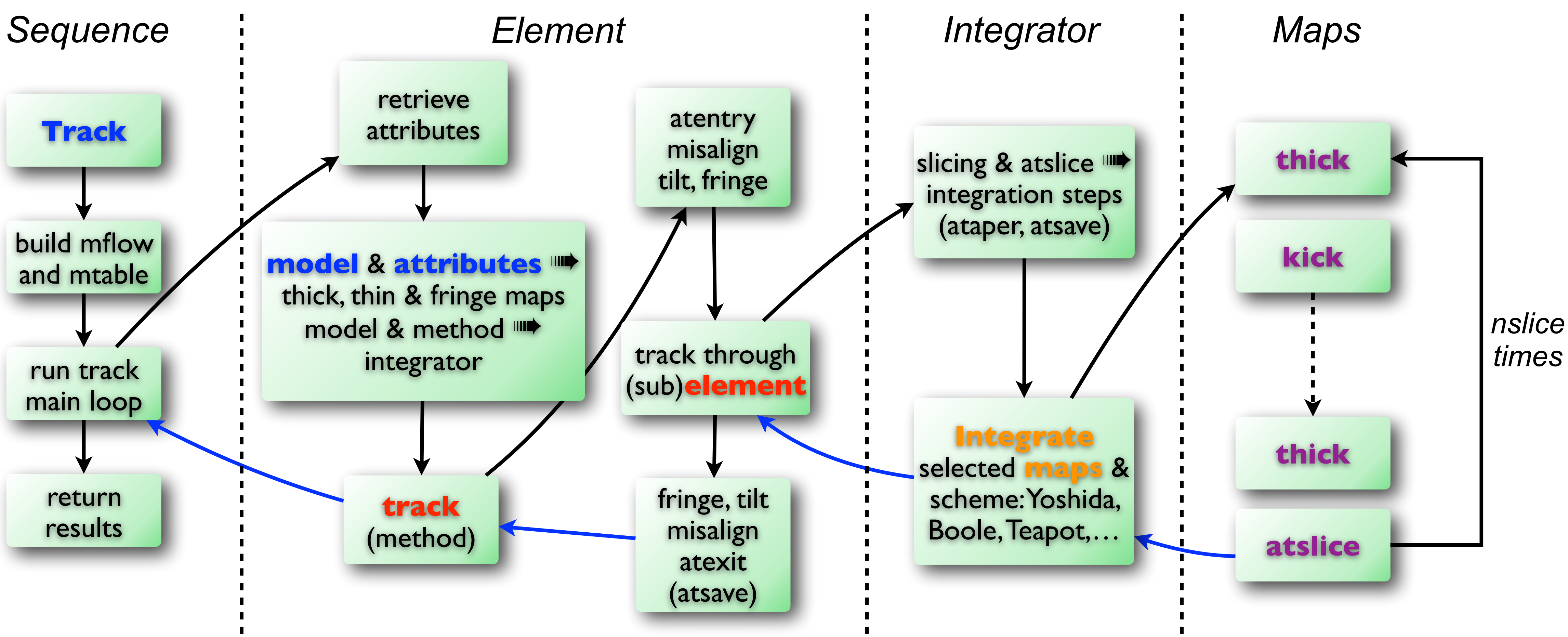


Physics can be parametrised and/or configured by element attributes and commands attributes

Physics can be extended by creating new element or modifying existing element or subelements track method (object oriented approach)

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Track in "depth" : user-defined possible extensions



Physics can be parametrised and/or configured by element attributes and commands attributes

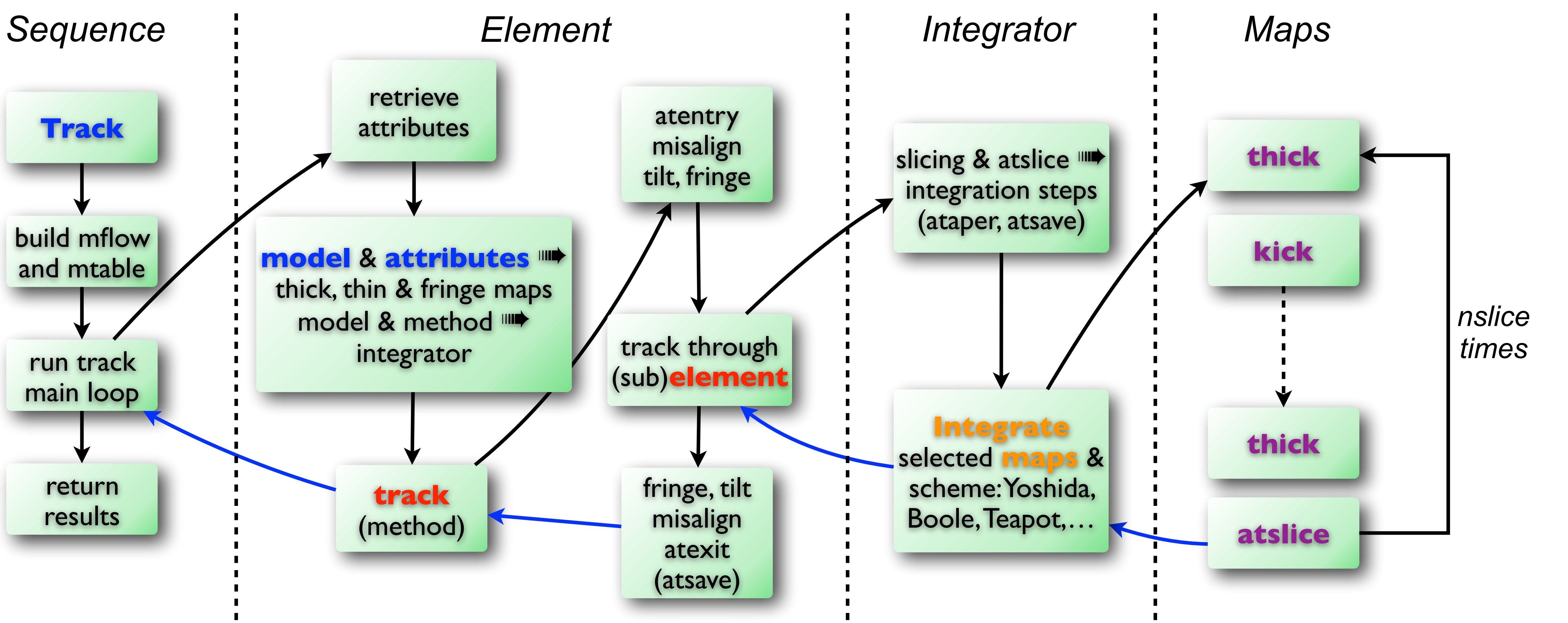
Physics can be extended by creating new element or modifying existing element or subelements track method (object oriented approach)

Physics can be extended by providing extra integration methods e.g. 3D field maps.

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Track in "depth" : user-defined possible extensions

BE Beams Department



Physics can be parametrised and/or configured by element attributes and commands attributes

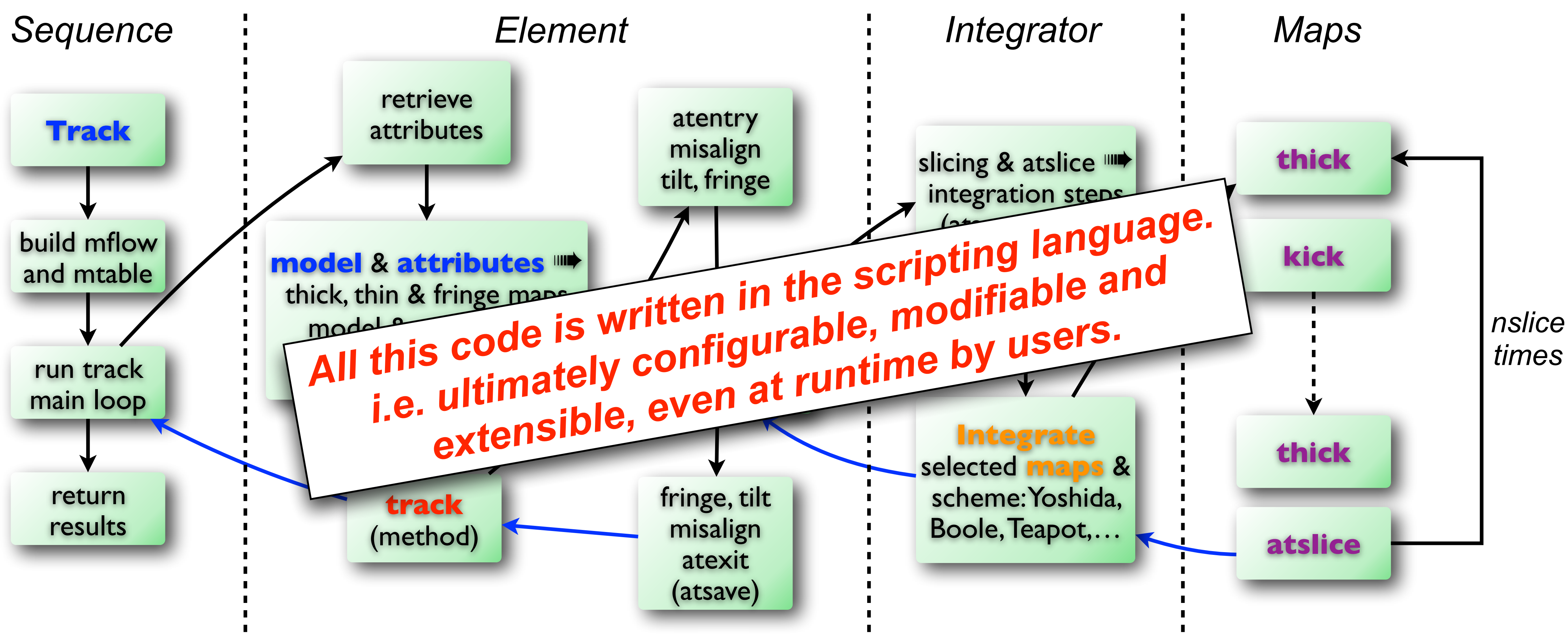
Physics can be extended by providing new maps or actions e.g. strong beam-beam (functional approach)

Physics can be extended by creating new element or modifying existing element or subelements track method (object oriented approach)

Physics can be extended by providing extra integration methods e.g. 3D field maps.

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Track in “depth” : user-defined possible extensions



Physics can be parametrised and/or configured by element attributes and commands attributes

Physics can be extended by providing new maps or actions e.g. strong beam-beam (functional approach)

Physics can be extended by creating new element or modifying existing element or subelements track method (object oriented approach)

Physics can be extended by providing extra integration methods e.g. 3D field maps.

MAD-NG physics I

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG physics I

- **6D PTC physics using GTPSA (for DA) and symplectic integrators.**
  - slicing, combined physics & elements, easy support for extensions, etc…
  - x4-10 faster than PTC for TPSA tracking, x1-2 slower than MAD-X for most cases.

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG physics I

- **6D PTC physics using GTPSA (for DA) and symplectic integrators.**
  - slicing, combined physics & elements, easy support for extensions, etc…
  - x4-10 faster than PTC for TPSA tracking, x1-2 slower than MAD-X for most cases.
- **Survey**: geometrical tracking
  - Survey supports multi-turns, ranged and step-by-step **backtracking** and **reverse** tracking. Return a Survey table *and* a Survey map flow (tracked context).
  - fully compatible with Track for superposition and observable points (e.g. table output, smooth plots, slicing, actions, sub-elements, etc…)
  - support exact misalignments and permanent misalignments, and patches.

MAD-NG physics I

- **6D PTC physics using GTPSA (for DA) and symplectic integrators.**
  - slicing, combined physics & elements, easy support for extensions, etc…
  - x4-10 faster than PTC for TPSA tracking, x1-2 slower than MAD-X for most cases.
- **Survey**: geometrical tracking
  - Survey supports multi-turns, ranged and step-by-step **backtracking** and **reverse** tracking. Return a Survey table *and* a Survey map flow (tracked context).
  - fully compatible with Track for superposition and observable points (e.g. table output, smooth plots, slicing, actions, sub-elements, etc…)
  - support exact misalignments and permanent misalignments, and patches.
- **Track**: dynamical tracking
  - Track supports **multi-particles** or **multi-damaps**, **multi-turns**, **ranged** and step-by-step **backtracking** and **reverse** tracking of **charged** particles to **arbitrary** DA **order** and arbitrary number of **parameters** (few thousands). Return a Track table *and* a Track map flow (tracked context).
  - fully compatible with Survey for superposition and observable points (same tracking engine).
  - support **exact** misalignments, **permanent** misalignments, **multipoles** & field errors **for all elements.** Can be combined freely with **patches**.
  - **symplectic tracking up to 8th order** on 5D (delta-p) and 6D (delta-rf) phase space *(exact=true, time=true, totalpath e.g. for thick RF)*.
  - provides true **thick lens** and thin lens tracking model, **radiation with photons tracking** (disabled in twiss), **fringe fields** (hard edge for all elements, including solenoid), **mutable particles** (multiple beams), **exact patches** (translations, rotations & time-energy), 4D weak-strong beam-beam (sixtracklib), apertures (all kinds).
  - may search for the closed orbit to support relative initial coordinates.

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG physics II

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG physics II

- **Cofind**: fix point search
  - Newton-based optimiser running **Track** with 1st order DA map or 7 particles.
  - support final coordinates translation.
  - extend Track with actions.

MAD-NG physics II

- **Cofind**: fix point search
  - Newton-based optimiser running **Track** with 1st order DA map or 7 particles.
  - support final coordinates translation.
  - extend Track with actions.
- **Twiss**: optics tracking
  - runs **Cofind** (closed orbit) - **Track** (one-turn map) - **Normal** - **Track** (optics) - post processing.
  - extend Track with actions to compute on-the-fly optics and fill twiss table (extended track table).
  - support coupled optics, dispersions, tunes, chromaticities, synchrotron integrals, momentum compaction factor, phase slip factor, energy gamma transition, etc… support *chrom* option to compute chromatic derivatives of previous quantities (e.g. Montaigue functions).

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG physics II

- **Cofind**: fix point search
  - Newton-based optimiser running **Track** with 1st order DA map or 7 particles.
  - support final coordinates translation.
  - extend Track with actions.
- **Twiss**: optics tracking
  - runs **Cofind** (closed orbit) - **Track** (one-turn map) - **Normal** - **Track** (optics) - post processing.
  - extend Track with actions to compute on-the-fly optics and fill twiss table (extended track table).
  - support coupled optics, dispersions, tunes, chromaticities, synchrotron integrals, momentum compaction factor, phase slip factor, energy gamma transition, etc… support *chrom* option to compute chromatic derivatives of previous quantities (e.g. Montaigue functions).
- **Match**: highly configurable optimiser
  - on the model of MAD-X `use_macro` approach, i.e. arbitrary user's setups & runs.
  - provides all kinds of local & global, linear & non-linear, optimiser (~20 algorithms).
  - very flexible, highly configurable with many physics-oriented setups (not just a penalty-function to minimise).

MAD-NG physics II

- **Cofind**: fix point search
  - Newton-based optimiser running **Track** with 1st order DA map or 7 particles.
  - support final coordinates translation.
  - extend Track with actions.
- **Twiss**: optics tracking
  - runs **Cofind** (closed orbit) - **Track** (one-turn map) - **Normal** - **Track** (optics) - post processing.
  - extend Track with actions to compute on-the-fly optics and fill twiss table (extended track table).
  - support coupled optics, dispersions, tunes, chromaticities, synchrotron integrals, momentum compaction factor, phase slip factor, energy gamma transition, etc… support *chrom* option to compute chromatic derivatives of previous quantities (e.g. Montaigue functions).
- **Match**: highly configurable optimiser
  - on the model of MAD-X `use_macro` approach, i.e. arbitrary user's setups & runs.
  - provides all kinds of local & global, linear & non-linear, optimiser (~20 algorithms).
  - very flexible, highly configurable with many physics-oriented setups (not just a penalty-function to minimise).
- **Correct**: orbit correction
  - provides few algorithms (e.g. SVD, Micado) to correct orbit using BPMs and Kickers. Supports many options.

MAD-NG physics II

- **Cofind**: fix point search
  - Newton-based optimiser running **Track** with 1st order DA map or 7 particles.
  - support final coordinates translation.
  - extend Track with actions.
- **Twiss**: optics tracking
  - runs **Cofind** (closed orbit) - **Track** (one-turn map) - **Normal** - **Track** (optics) - post processing.
  - extend Track with actions to compute on-the-fly optics and fill twiss table (extended track table).
  - support coupled optics, dispersions, tunes, chromaticities, synchrotron integrals, momentum compaction factor, phase slip factor, energy gamma transition, etc… support *chrom* option to compute chromatic derivatives of previous quantities (e.g. Montaigue functions).
- **Match**: highly configurable optimiser
  - on the model of MAD-X `use_macro` approach, i.e. arbitrary user's setups & runs.
  - provides all kinds of local & global, linear & non-linear, optimiser (~20 algorithms).
  - very flexible, highly configurable with many physics-oriented setups (not just a penalty-function to minimise).
- **Correct**: orbit correction
  - provides few algorithms (e.g. SVD, Micado) to correct orbit using BPMs and Kickers. Supports many options.
- **Normal**: normal forms analysis *(under validation)*
  - provides linear and non-linear parametric normal forms on DA map (used by twiss) to extract RDTs. Can be applied at observable points in Track to track RDTs, either on-the-fly with actions or through post processing of DA maps saved in Track table.

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG review

- Performed from Oct. 2020 to Mar. 2021.

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG review

- Performed from Oct. 2020 to Mar. 2021.
- Run **simple studies** on CERN machines and compare results vs MAD-X and MADX-PTC (listed in reverse time order, from last to first).
  - ➡ *Clic 380 GeV BDS optimisation* (Andrii Pastushenko, 2 presentations)
    - twiss, high order maps generation, beam size comparison.
  - ➡ *MAD-NG outlook for LHC and HL-LHC* (Riccardo De Maria)
  - ➡ *MAD-NG in Gantries* (Cedric Hernalsteens, not presented)
  - ➡ *Experience with FCC-ee Lattice in MAD-NG* (Leon van Riesen-Haupt)
    - linear optics, momentum detuning, amplitude detuning, radiation integrals.
  - ➡ *Experience for LHC coupling with MAD-NG* (Tobias Persson).
    - example in the next slide
  - ➡ *Experience of MAD-NG* with the PS (Alexander Huschauer).
    - linear optics, dispersions, tunes, chromaticities.
    - exploration of model and integration methods.
  - ➡ *Translating MAD-X scripts to MAD-NG* (Laurent Deniau).

MAD-NG studies - LHC coupling with param. maps

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG studies - LHC coupling with param. maps

```
print("strengths before matching coupling correctors:")
print("sk1r=", MADX.sk1r)
print("sk2r=", MADX.sk2r)
print("sk3r=", MADX.sk3r)
print("sk4r=", MADX.sk4r)

local X0 = damap {mo=2, nv=6, nk=4, ko=1,
                  vn={'x','px','y','py','t','pt',
                      'sk1r','sk2r','sk3r','sk4r'}}

-- set knobs: scalar + TPSA -> TPSA
MADX.sk1r = MADX.sk1r + X0.sk1r
MADX.sk2r = MADX.sk2r + X0.sk2r
MADX.sk3r = MADX.sk3r + X0.sk3r
MADX.sk4r = MADX.sk4r + X0.sk4r

local mjac = { ---> variables & knobs
  { var='x' ,'0010001','00100001','001000001','0010000001' }, --    |
  { var='x' ,'0001001','00010001','000100001','0001000001' }, --    |
  { var='px','0010001','00100001','001000001','0010000001' }, --    v
  { var='px','0001001','00010001','000100001','0001000001' }, -- constraints
}

status, fmin, ncall = match {
  command := track {sequence=lhcb1, X0=X0, observe=1, savemap=true},
…
```

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG studies - LHC coupling with param. maps

BE Beams Department

```
print("strengths before matching coupling correctors:")
print("sk1r=", MADX.sk1r)
print("sk2r=", MADX.sk2r)
print("sk3r=", MADX.sk3r)
print("sk4r=", MADX.sk4r)

local X0 = damap {mo=2, nv=6, nk=4, ko=1,
                  vn={'x','px','y','py','t','pt',
                      'sk1r','sk2r','sk3r','sk4r'}}

-- set knobs: scalar + TPSA -> TPSA
MADX.sk1r = MADX.sk1r + X0.sk1r
MADX.sk2r = MADX.sk2r + X0.sk2r
MADX.sk3r = MADX.sk3r + X0.sk3r
MADX.sk4r = MADX.sk4r + X0.sk4r

local mjac = { ---> variables & knobs
  { var='x' ,'0010001','00100001','001000001','001000
  { var='x' ,'0001001','00010001','000100001','000100
  { var='px','0010001','00100001','001000001','001000
  { var='px','0001001','00010001','000100001','000100
}

status, fmin, ncall = match {
  command := track {sequence=lhcb1, X0=X0, observe=1,
…
```

```
status, fmin, ncall = match {
  command := track {sequence=lhcb1, X0=X0, observe=1, savemap=true},

  jacobian = \t,grd,jac => -- gradient not used, fill only jacobian
    jac:setrow(1.. 8, t['S.DS.L2.B1'].__map:getm(mjac) )
    jac:setrow(9..16, t['E.DS.L2.B1'].__map:getm(mjac) )
  end,

  variables = { rtol=1e-6, -- 1 ppm
    { name='sk1r', get := MADX.sk1r:get0(), set = \x -> MADX.sk1r:set0(x) },
    { name='sk2r', get := MADX.sk2r:get0(), set = \x -> MADX.sk2r:set0(x) },
    { name='sk3r', get := MADX.sk3r:get0(), set = \x -> MADX.sk3r:set0(x) },
    { name='sk4r', get := MADX.sk4r:get0(), set = \x -> MADX.sk4r:set0(x) },
  },

  equalities = {
    { expr = \t -> t['S.DS.L2.B1'].__map.x :get'0010', name='S.R11.x' },
    { expr = \t -> t['S.DS.L2.B1'].__map.x :get'0001', name='S.R12.x' },
    { expr = \t -> t['S.DS.L2.B1'].__map.px:get'0010', name='S.R21.x' },
    { expr = \t -> t['S.DS.L2.B1'].__map.px:get'0001', name='S.R22.x' },

    { expr = \t -> t['E.DS.L2.B1'].__map.x :get'0010', name='E.R11.x' },
    { expr = \t -> t['E.DS.L2.B1'].__map.x :get'0001', name='E.R12.x' },
    { expr = \t -> t['E.DS.L2.B1'].__map.px:get'0010', name='E.R21.x' },
    { expr = \t -> t['E.DS.L2.B1'].__map.px:get'0001', name='E.R22.x' },
 },
  objective = { fmin=1e-12 },
  maxcall=100, info=2
}

-- reset knobs: extract scalar values from TPSA
MADX.sk1r = MADX.sk1r:get0()
MADX.sk2r = MADX.sk2r:get0()
MADX.sk3r = MADX.sk3r:get0()
MADX.sk4r = MADX.sk4r:get0()

print("status=", status, "fmin=", fmin, "ncall=", ncall)
print("strengths after matching coupling correctors:")
print("sk1r=" , MADX.sk1r )
print("sk2r=" , MADX.sk2r )
print("sk3r=" , MADX.sk3r )
print("sk4r=" , MADX.sk4r )
```

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

MAD-NG studies - LHC coupling with param. maps

```
print("strengths before matching coupling correctors:")
print("sk1r=", MADX.sk1r)
print("sk2r=", MADX.sk2r)
print("sk3r=", MADX.sk3r)
print("sk4r=", MADX.sk4r)

local X0 = damap {mo=2, nv=6, nk=4, ko=1,
                  vn={'x','px','y','py','t','pt',
                      'sk1r','sk2r','sk3r','sk4r'}}

-- set knobs: scalar + TPSA -> TPSA
MADX.sk1r = MADX.sk1r + X0.sk1r
MADX.sk2r = MADX.sk2r + X0.sk2r
MADX.sk3r = MADX.sk3r + X0.sk3r
MADX.sk4r = MADX.sk4r + X0.sk4r

local mjac = { ---> variables & knobs
  { var='x' ,'0010001','00100001','001000001','001000
  { var='x' ,'0001001','00010001','000100001','000100
  { var='px','0010001','00100001','001000001','001000
  { var='px','0001001','00010001','000100001','000100
}

status, fmin, ncall = match {
  command := track {sequence=lhcb1, X0=X0, observe=1,
…
```

```
status, fmin, ncall = match {
  command := track {sequence=lhcb1, X0=X0, observe=1, savemap=true},

  jacobian = \t,grd,jac => -- gradient not used, fill only jacobian
    jac:setrow(1.. 8, t['S.DS.L2.B1'].__map:getm(mjac) )
    jac:setrow(9..16, t['E.DS.L2.B1'].__map:getm(mjac) )
  end,

  variables = { rtol=1e-6, -- 1 ppm
    { name='sk1r', get := MADX.sk1r:get0(), set = \x -> MADX.sk1r:set0(x) },
    { name='sk2r', get := MADX.sk2r:get0(), set = \x -> MADX.sk2r:set0(x) },
    { name='sk3r', get := MADX.sk3r:get0(), set = \x -> MADX.sk3r:set0(x) },
    { name='sk4r', get := MADX.sk4r:get0(), set = \x -> MADX.sk4r:set0(x) },
  },

  equalities = {
    { expr = \t -> t['S.DS.L2.B1'].__map.x :get'0010', name='S.R11.x' },
    { expr = \t -> t['S.DS.L2.B1'].__map.x :get'0001', name='S.R12.x' },
    { expr = \t -> t['S.DS.L2.B1'].__map.px:get'0010', name='S.R21.x' },
    { expr = \t -> t['S.DS.L2.B1'].__map.px:get'0001', name='S.R22.x' },

    { expr = \t -> t['E.DS.L2.B1'].__map.x :get'0010', name='E.R11.x' },
    { expr = \t -> t['E.DS.L2.B1'].__map.x :get'0001', name='E.R12.x' },
    { expr = \t -> t['E.DS.L2.B1'].__map.px:get'0010', name='E.R21.x' },
    { expr = \t -> t['E.DS.L2.B1'].__map.px:get'0001', name='E.R22.x' },
 },
  objective = { fmin=1e-12 },
  maxcall=100, info=2
}

-- reset knobs: extract scalar values from TPSA
MADX.sk1r = MADX.sk1r:get0()
MADX.sk2r = MADX.sk2r:get0()
MADX.sk3r = MADX.sk3r:get0()
MADX.sk4r = MADX.sk4r:get0()

print("status=", status, "fmin=", fmin, "ncall=", ncall)
print("strengths after matching coupling correctors:")
print("sk1r=" , MADX.sk1r )
print("sk2r=" , MADX.sk2r )
print("sk3r=" , MADX.sk3r )
print("sk4r=" , MADX.sk4r )
```

Timing summary and links to codes:

MAD-X using matrix 1m55
MAD-NG using matrix 55s (15s)
MAD-NG using matrix & knobs 40s (4.5s)
MADX-PTC using alphas-betas >40m

MAD-NG studies - LHC coupling with param. maps

```
print("strengths before matching coupling correctors:")
print("sk1r=", MADX.sk1r)
print("sk2r=", MADX.sk2r)
print("sk3r=", MADX.sk3r)
print("sk4r=", MADX.sk4r)

local X0 = damap {mo=2, nv=6, nk=4, ko=1,
                  vn={'x','px','y','py','t','pt',
                      'sk1r','sk2r','sk3r','sk4r'}}

-- set knobs: scalar + TPSA -> TPSA
MADX.sk1r = MADX.sk1r + X0.sk1r
MADX.sk2r = MADX.sk2r + X0.sk2r
MADX.sk3r = MADX.sk3r + X0.sk3r
MADX.sk4r = MADX.sk4r + X0.sk4r

local mjac = { ---> variables & knobs
  { var='x' ,'0010001','00100001','001000001','001000
  { var='x' ,'0001001','00010001','000100001','000100
  { var='px','0010001','00100001','001000001','001000
  { var='px','0001001','00010001','000100001','000100
}

status, fmin, ncall = match {
  command := track {sequence=lhcb1, X0=X0, observe=1,
…
```

```
status, fmin, ncall = match {
  command := track {sequence=lhcb1, X0=X0, observe=1, savemap=true},

  jacobian = \t,grd,jac => -- gradient not used, fill only jacobian
    jac:setrow(1.. 8, t['S.DS.L2.B1'].__map:getm(mjac) )
    jac:setrow(9..16, t['E.DS.L2.B1'].__map:getm(mjac) )
  end,

  variables = { rtol=1e-6, -- 1 ppm
    { name='sk1r', get := MADX.sk1r:get0(), set = \x -> MADX.sk1r:set0(x) },
    { name='sk2r', get := MADX.sk2r:get0(), set = \x -> MADX.sk2r:set0(x) },
    { name='sk3r', get := MADX.sk3r:get0(), set = \x -> MADX.sk3r:set0(x) },
    { name='sk4r', get := MADX.sk4r:get0(), set = \x -> MADX.sk4r:set0(x) },
  },

  equalities = {
    { expr = \t -> t['S.DS.L2.B1'].__map.x :get'0010', name='S.R11.x' },
    { expr = \t -> t['S.DS.L2.B1'].__map.x :get'0001', name='S.R12.x' },
    { expr = \t -> t['S.DS.L2.B1'].__map.px:get'0010', name='S.R21.x' },
    { expr = \t -> t['S.DS.L2.B1'].__map.px:get'0001', name='S.R22.x' },

    { expr = \t -> t['E.DS.L2.B1'].__map.x :get'0010', name='E.R11.x' },
    { expr = \t -> t['E.DS.L2.B1'].__map.x :get'0001', name='E.R12.x' },
    { expr = \t -> t['E.DS.L2.B1'].__map.px:get'0010', name='E.R21.x' },
    { expr = \t -> t['E.DS.L2.B1'].__map.px:get'0001', name='E.R22.x' },
  },
  objective = { fmin=1e-12 },
  maxcall=100, info=2

eset knobs: extract scalar values from TPSA
.sk1r = MADX.sk1r:get0()
.sk2r = MADX.sk2r:get0()
sk3r = MADX.sk3r:get0()
sk4r = MADX.sk4r:get0()

"status=", status, "fmin=", fmin, "ncall=", ncall)
"strengths after matching coupling correctors:")
print("sk1r=" , MADX.sk1r )
print("sk2r=" , MADX.sk2r )
print("sk3r=" , MADX.sk3r )
print("sk4r=" , MADX.sk4r )
```

Timing summary and links to codes:
MAD-X using matrix 1m55
MAD-NG using matrix 55s (15s)
MAD-NG using matrix & knobs 4…
MADX-PTC usi…

Match command performs a Principal Component Analysis on the Jacobian and tags useless constraints and variables, i.e. starting with oversized set of knobs or constrains does not harm when using parametric maps!

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Conclusions

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Conclusions

Beams Department

- MAD-NG is reaching the end of its development process.

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Conclusions

- MAD-NG is reaching the end of its development process.
- 2022 will focus on participation to real studies and consolidation.
  - bottom-top validation for the physics of real case studies.
  - add missing physics on demand (e.g. tapering, spin, generalised multipoles).
  - complete unit tests & manual.
  - improve performance (room for x3-x5 in speed).
  - simplify some aspects, “simpler is better” (e.g. object model).

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Conclusions

- MAD-NG is reaching the end of its development process.
- 2022 will focus on participation to real studies and consolidation.
  - bottom-top validation for the physics of real case studies.
  - add missing physics on demand (e.g. tapering, spin, generalised multipoles).
  - complete unit tests & manual.
  - improve performance (room for x3-x5 in speed).
  - simplify some aspects, “simpler is better” (e.g. object model).
- On some aspects, MAD-NG is more mature than MAD-X
  - better code architecture and structure.
  - more flexible and extensible for the physics (new features require day(s)).
  - less surprises when combining features (e.g. misalignments and slicing).
  - main stream programming language for scripting (save user time!) & many toolboxes.
  - mature technologies, syntax error, backtrace, debugger, profiler, JIT (save user time!).
  - some features have been back ported to MAD-X (e.g. permanent misalignment, patches) or will be (fringe fields, combined/overlapping elements).
  - support backtracking, charged particles, parallel sequences, useful for e.g. matching IPs, no need for reverse sequence, etc…

Conclusions

- MAD-NG is reaching the end of its development process.
- 2022 will focus on participation to real studies and consolidation.
  - bottom-top validation for the physics of real case studies.
  - add missing physics on demand (e.g. tapering, spin, generalised multipoles).
  - complete unit tests & manual.
  - improve performance (room for x3-x5 in speed).
  - simplify some aspects, “simpler is better” (e.g. object model).
- On some aspects, MAD-NG is more mature than MAD-X
  - better code architecture and structure.
  - more flexible and extensible for the physics (new features require day(s)).
  - less surprises when combining features (e.g. misalignments and slicing).
  - main stream programming language for scripting (save user time!) & many toolboxes.
  - mature technologies, syntax error, backtrace, debugger, profiler, JIT (save user time!).
  - some features have been back ported to MAD-X (e.g. permanent misalignment, patches) or will be (fringe fields, combined/overlapping elements).
  - support backtracking, charged particles, parallel sequences, useful for e.g. matching IPs, no need for reverse sequence, etc…

Do not hesitate to ask me some help!

CERN

BE Beams Department

THANK YOU FOR YOUR ATTENTION

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Lua overview (http://www.lua.org)

Beams Department

MAD scripting language is based on Lua 5.1+ (it is a superset of)

- about
- news
- get started
- download
- documentation
- community
- contact
- site map
- português

Lua 5.3.3
released

Programando em Lua
published

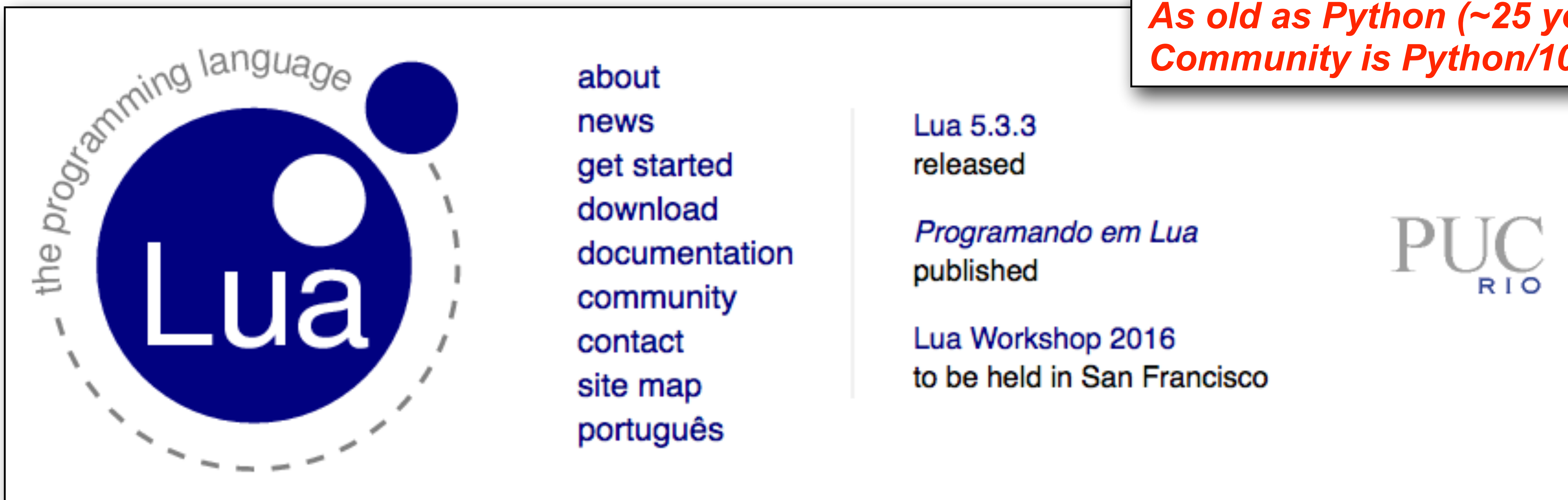
Lua Workshop 2016
to be held in San Francisco

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Lua overview (http://www.lua.org)

MAD scripting language is based on Lua 5.1+ (it is a superset of)

As old as Python (~25 years)
Community is Python/10



Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Lua overview (http://www.lua.org)

MAD scripting language is based on Lua 5.1+ (it is a superset of)

As old as Python (~25 years)
Community is Python/10

the programming language Lua

- about
- news
- get started
- download
- documentation
- community
- contact
- site map
- português

Lua 5.3.3 released

Programando em Lua published

Lua Workshop 2016 to be held in San Francisco

PUC RIO

Lua is a powerful, fast, lightweight, embeddable scripting language.

Lua combines simple procedural syntax with powerful data description constructs based on **associative arrays** and **extensible semantics**. Lua is **dynamically typed**, runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it **ideal for configuration, scripting, and rapid prototyping**.

Lua has been used in many industrial applications (e.g., Adobe's Photoshop Lightroom), with an emphasis on embedded systems (e.g., the Ginga middleware for digital TV in Brazil) and games (e.g., World of Warcraft and Angry Birds). Lua is currently the leading scripting language in games. **Lua has a solid reference manual and there are several books about it.** Several versions of Lua have been released and used in real applications since its **creation in 1993**. Lua featured in HOPL III, the Third ACM SIGPLAN History of Programming Languages Conference, in June 2007. Lua won the Front Line Award 2011 from the Game Developers Magazine.

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Lua overview (http://www.lua.org)

MAD scripting language is based on Lua 5.1+ (it is a superset of)

As old as Python (~25 years)
Community is Python/10

the programming language Lua

- about
- news
- get started
- download
- documentation
- community
- contact
- site map
- português

Lua 5.3.3
released

Programando em Lua
published

Lua Workshop 2016
to be held in San Francisco

PUC RIO

Reference manual is 29 pages!

Lua is a powerful, fast, lightweight, embeddable scripting language.

Lua combines simple procedural syntax with powerful data description constructs based on **associative arrays** and **extensible semantics**. Lua is **dynamically typed**, runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it **ideal for configuration, scripting, and rapid prototyping**.

Lua has been used in many industrial applications (e.g., Adobe's Photoshop Lightroom), with an emphasis on embedded systems (e.g., the Ginga middleware for digital TV in Brazil) and games (e.g., World of Warcraft and Angry Birds). Lua is currently the leading scripting language in games. **Lua has a solid reference manual and there are several books about it.** Several versions of Lua have been released and used in real applications since its **creation in 1993**. Lua featured in HOPL III, the Third ACM SIGPLAN History of Programming Languages Conference, in June 2007. Lua won the Front Line Award 2011 from the Game Developers Magazine.

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

LuaJIT overview (http://www.luajit.org)

LuaJIT is a **Just-In-Time Compiler** (JIT) for the Lua programming language. Lua is a powerful, dynamic and light-weight programming language. It may be embedded or used as a general-purpose, stand-alone language.

Compatibility

Windows	Linux	BSD	OSX	POSIX	
Embedded	Android	iOS			
PS3	PS4	PS Vita	Xbox 360		
GCC	CLANG LLVM	MSVC			
x86	x64	ARM	PPC	e500	MIPS
Lua 5.1 API+ABI	+ JIT	+ BitOp	+ FFI	Drop-in DLL/.so	

Overview

3x - 100x	115 KB VM	90 KB JIT	63 KLOC C	24 KLOC ASM	11 KLOC Lua

LuaJIT has been successfully used as a **scripting middleware** in games, appliances, network and graphics apps, numerical simulations, trading platforms and many other specialty applications. It scales from embedded devices, smartphones, desktops up to server farms. It combines high flexibility with high performance and an unmatched **low memory footprint**.

LuaJIT has been in continuous development since 2005. It's widely considered to be **one of the fastest dynamic language implementations**. It has outperformed other dynamic languages on many cross-language benchmarks since its first release — often by a substantial margin. For **LuaJIT 2.0**, the whole VM has been rewritten from the ground up and relentlessly optimised for performance. It combines a **high-speed interpreter**, written in assembler, with a **state-of-the-art JIT compiler**.

An innovative **trace compiler** is integrated with advanced, SSA-based optimisations and highly tuned code generation backends. A substantial reduction of the overhead associated with dynamic languages allows it **to break into the performance range traditionally reserved for offline, static language compilers.**

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

LuaJIT overview (http://www.luajit.org)

BE Beams Department

LuaJIT is a **Just-In-Time Compiler** (JIT) for the Lua programming language. Lua is a powerful, dynamic and light-weight programming language. It may be embedded or used as a general-purpose, stand-alone language.

Compatibility

Windows	Linux	BSD	OSX	POSIX	
Embedded	Android	iOS			
PS3	PS4	PS Vita	Xbox 360		
GCC	CLANG LLVM	MSVC			
x86	x64	ARM	PPC	e500	MIPS
Lua 5.1 API+ABI	+ JIT	+ BitOp	+ FFI	Drop-in DLL/.so	

Overview

3x - 100x	115 KB VM	90 KB JIT	63 KLOC C	24 KLOC ASM	11 KLOC Lua

LuaJIT has been successfully used as a **scripting middleware** in games, appliances, network and graphics apps, numerical simulations, trading platforms and many other specialty applications. It scales from embedded devices, smartphones, desktops up to server farms. It combines high flexibility with high performance and an unmatched **low memory footprint**.

LuaJIT has been in continuous development since 2005. It's widely considered to be **one of the fastest dynamic language implementations**. It has outperformed other dynamic languages on many cross-language benchmarks since its first release — often by a substantial margin.
For **LuaJIT 2.0**, the whole VM has been rewritten from the ground up and relentlessly optimised for performance. It combines a **high-speed interpreter**, written in assembler, with a **state-of-the-art JIT compiler**.

An innovative **trace compiler** is integrated with advanced, SSA-based optimisations and highly tuned code generation backends. A substantial reduction of the overhead associated with dynamic languages allows it **to break into the performance range traditionally reserved for offline, static language compilers.**

From M. Pall website, author of LuaJIT

LuaJIT overview (http://www.luajit.org)

LuaJIT is a **Just-In-Time Compiler** (JIT) for the Lua programming language. Lua is a powerful, dynamic and light-weight programming language. It may be embedded or used as a general-purpose, stand-alone language.

LuaJIT is Copyright © 2005-2015 Mike Pall, released under the MIT open source license.

Compatibility

Windows	Linux	BSD	OSX	POSIX	
Embedded	Android	iOS			
PS3	PS4	PS Vita	Xbox 360		
GCC	CLANG LLVM	MSVC			
x86	x64	ARM	PPC	e500	MIPS
Lua 5.1 API+ABI	+ JIT	+ BitOp	+ FFI	Drop-in DLL/.so	

Overview

3x - 100x	115 KB VM	90 KB JIT	63 KLOC C	24 KLOC ASM	11 KLOC Lua

From M. Pall website, author of LuaJIT

LuaJIT has been successfully used as a **scripting middleware** in games, appliances, network and graphics apps, numerical simulations, trading platforms and many other specialty applications. It scales from embedded devices, smartphones, desktops up to server farms. It combines high flexibility with high performance and an unmatched **low memory footprint**.

LuaJIT has been in continuous development since 2005. It's widely considered to be **one of the fastest dynamic language implementations**. It has outperformed other dynamic languages on many cross-language benchmarks since its first release — often by a substantial margin.
For **LuaJIT 2.0**, the whole VM has been rewritten from the ground up and relentlessly optimised for performance. It combines a **high-speed interpreter**, written in assembler, with a **state-of-the-art JIT compiler**.

An innovative **trace compiler** is integrated with advanced, SSA-based optimisations and highly tuned code generation backends. A substantial reduction of the overhead associated with dynamic languages allows it **to break into the performance range traditionally reserved for offline, static language compilers.**

As old as PyPy (~10 years)
Community is ~PyPy

LuaJIT overview (http://www.luajit.org)

Beams Department

LuaJIT is a **Just-In-Time Compiler** (JIT) for the Lua programming language. Lua is a powerful, dynamic and light-weight programming language. It may be embedded or used as a general-purpose, stand-alone language.

LuaJIT is Copyright © 2005-2015 Mike Pall, released under the MIT open source license.

Compatibility

Windows	Linux	BSD	OSX	POSIX	
Embedded	Android	iOS			
PS3	PS4	PS Vita	Xbox 360		
GCC	CLANG LLVM	MSVC			
x86	x64	ARM	PPC	e500	MIPS
Lua 5.1 API+ABI	+ JIT	+ BitOp	+ FFI	Drop-in DLL/.so	

Overview

3x - 100x	115 KB VM	90 KB JIT	63 KLOC C	24 KLOC ASM	11 KLOC Lua

From M. Pall website, author of LuaJIT

LuaJIT has been successfully used as a **scripting middleware** in games, appliances, network and graphics apps, numerical simulations, trading platforms and many other specialty applications. It scales from embedded devices, smartphones, desktops up to server farms. It combines high flexibility with high performance and an unmatched **low memory footprint**.

LuaJIT has been in continuous development since 2005. It's widely considered to be **one of the fastest dynamic language implementations**. It has outperformed other dynamic languages on many cross-language benchmarks since its first release — often by a substantial margin.

For **LuaJIT 2.0**, the whole VM has been rewritten from the ground up and relentlessly optimised for performance. It combines a **high-speed interpreter**, written in assembler, with a **state-of-the-art JIT compiler**.

An innovative **trace compiler** is integrated with advanced, SSA-based optimisations and highly tuned code generation backends. A substantial reduction of the overhead associated with dynamic languages allows it **to break into the performance range traditionally reserved for offline, static language compilers.**

As old as PyPy (~10 years)
Community is ~PyPy

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

Beams Department

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \frac{f_a^{(k)}}{k!}(x-a)^k$$

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f :

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

$f(x)$ is an analytic function, $T_f^n(x;a)$ is a polynomial approximation nearby a with radius of convergence h: $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

$f(x)$ is an analytic function, $T_f^n(x;a)$ is a polynomial approximation nearby a with radius of convergence h: $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables *(x,y)* at order 2 nearby *(a,b)*:

$$T_f^2(x,y;a,b) = f(a,b) + \left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b) + \ldots$$

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in ℝ & ℂ.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

$f(x)$ is an analytic function, $T_f^n(x;a)$ is a polynomial approximation nearby a with radius of convergence h: $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables *(x,y)* at order 2 nearby *(a,b)*:

$$= f_{(a,b)}^{(1)}(x-a, y-b)$$

$$T_f^2(x,y;a,b) = f(a,b) + \boxed{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)} + \ldots$$

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

f(x) is an analytic function, $T_f^n(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence h:* $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables *(x,y)* at order 2 nearby *(a,b)*:

$$= f_{(a,b)}^{(1)}(x-a, y-b)$$

$$T_f^2(x,y;a,b) = f(a,b) + \boxed{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)} + \ldots$$

$$+ \frac{1}{2!}\left(\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + 2\left.\frac{\partial^2 f}{\partial x \partial y}\right|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2 \right)$$

GTPSA in a nutshell

BE Beams Department

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

f(x) is an analytic function, $T_f^n(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence h:* $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables *(x,y)* at order 2 nearby *(a,b)*:

$$= f^{(1)}_{(a,b)}(x-a, y-b)$$

$$T_f^2(x,y;a,b) = f(a,b) + \boxed{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)} + \ldots$$

$$+\frac{1}{2!}\left(\boxed{\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + 2\left.\frac{\partial^2 f}{\partial x \partial y}\right|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2}\right)$$

$$= f^{(2)}_{(a,b)}(x-a, y-b)$$

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in ℝ & ℂ.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

f(x) is an analytic function, $T_f^n(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence* h*:* $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables (x,y) at order 2 nearby (a,b):

$$= f_{(a,b)}^{(1)}(x-a, y-b)$$

$$T_f^2(x,y;a,b) = f(a,b) + \boxed{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)} + \ldots$$

homogeneous polynomials

$$+\frac{1}{2!}\left(\boxed{\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + 2\left.\frac{\partial^2 f}{\partial x \partial y}\right|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2}\right)$$

$$= f_{(a,b)}^{(2)}(x-a, y-b)$$

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in ℝ & ℂ.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

f(x) is an analytic function, $T_f^n(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence h:* $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables *(x,y)* at order 2 nearby *(a,b)*:

$$T_f^2(x,y;a,b) = f(a,b) + \underbrace{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)}_{= f^{(1)}_{(a,b)}(x-a,\, y-b)} + \ldots$$

$$+\frac{1}{2!}\Bigg(\underbrace{\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + \boxed{2\frac{\partial^2 f}{\partial x \partial y}}\Bigg|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2}_{= f^{(2)}_{(a,b)}(x-a,\, y-b)}\Bigg)$$

homogeneous polynomials

f must not depend on the integration path, i.e. must derive from a potential!

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in ℝ & ℂ.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

f(x) is an analytic function, $T_f^n(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence h:* $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables *(x,y)* at order 2 nearby *(a,b)*:

$$= f_{(a,b)}^{(1)}(x-a, y-b)$$

$$T_f^2(x,y;a,b) = f(a,b) + \boxed{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)} + \ldots$$

homogeneous polynomials

f must not depend on the integration path, i.e. must derive from a potential!

$$+\frac{1}{2!}\left(\boxed{\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + \left.\boxed{2\frac{\partial^2 f}{\partial x \partial y}}\right|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2}\right)$$

$$= f_{(a,b)}^{(2)}(x-a, y-b)$$

v variables X at order n nearby A:

$$T_f^n(X;A) = \sum_{k=0}^{n} \frac{f_A^{(k)}}{k!}(X;A)^k = \sum_{k=0}^{n} \frac{1}{k!} \sum_{|\vec{m}|=k} \binom{k}{\vec{m}} \left.\frac{\partial^k f}{\partial X^{\vec{m}}}\right|_A (X;A)^{\vec{m}} \quad \text{with} \quad \binom{k}{\vec{m}} = \frac{k!}{c_1!\,c_2!\ldots c_v!}$$

GTPSA in a nutshell

Beams Department

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

$f(x)$ is an analytic function, $T_f^n(x;a)$ is a polynomial approximation nearby a with radius of convergence h: $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables *(x,y)* at order 2 nearby *(a,b)*:

$$= f_{(a,b)}^{(1)}(x-a, y-b)$$

$$T_f^2(x,y;a,b) = f(a,b) + \boxed{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)} + \ldots$$

homogeneous polynomials

f must not depend on the integration path, i.e. must derive from a potential!

$$+\frac{1}{2!}\left(\boxed{\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + \boxed{2\frac{\partial^2 f}{\partial x \partial y}}\right|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2}\right)$$

$$= f_{(a,b)}^{(2)}(x-a, y-b)$$

v variables X at order n nearby A:

$$T_f^n(X;A) = \sum_{k=0}^{n} \frac{f_A^{(k)}}{k!}(X;A)^k = \sum_{k=0}^{n} \frac{1}{k!} \sum_{\boxed{|\vec{m}|=k}} \binom{k}{\vec{m}} \left.\frac{\partial^k f}{\partial X^{\vec{m}}}\right|_A (X;A)^{\vec{m}} \quad \text{with} \quad \binom{k}{\vec{m}} = \frac{k!}{c_1!\,c_2!\ldots c_v!}$$

monomials of order k

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in ℝ & ℂ.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

f(x) is an analytic function, $T_f^n(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence h:* $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables (x,y) at order 2 nearby (a,b):

$$T_f^2(x,y;a,b) = f(a,b) + \underbrace{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)}_{= f^{(1)}_{(a,b)}(x-a,\,y-b)} + \ldots$$

$$+\frac{1}{2!}\Bigg(\underbrace{\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + \boxed{2\frac{\partial^2 f}{\partial x \partial y}}\Bigg|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2}_{= f^{(2)}_{(a,b)}(x-a,\,y-b)}\Bigg)$$

homogeneous polynomials

f must not depend on the integration path, i.e. must derive from a potential!

v variables X at order n nearby A:

$$T_f^n(X;A) = \sum_{k=0}^{n} \frac{f_A^{(k)}}{k!}(X;A)^k = \sum_{k=0}^{n} \frac{1}{k!} \sum_{\boxed{|\vec{m}|=k}} \binom{k}{\vec{m}} \left.\frac{\partial^k f}{\partial X^{\vec{m}}}\right|_A (X;A)^{\vec{m}} \quad \text{with} \quad \boxed{\binom{k}{\vec{m}}} = \frac{k!}{c_1!\,c_2!\ldots c_v!}$$

monomials of order k

multinomial

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra
  - ➡ Multivariate Taylor polynomials of order n in ℝ & ℂ.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

f(x) is an analytic function, $T_f^n(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence h:* $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables (x,y) at order 2 nearby (a,b):

$$= f_{(a,b)}^{(1)}(x-a, y-b)$$

$$T_f^2(x,y;a,b) = f(a,b) + \boxed{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)} + \ldots$$

homogeneous polynomials

f must not depend on the integration path, i.e. must derive from a potential!

$$+\frac{1}{2!}\left(\boxed{\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + \boxed{2\frac{\partial^2 f}{\partial x \partial y}}\bigg|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2}\right)$$

$$= f_{(a,b)}^{(2)}(x-a, y-b)$$

v variables X at order n nearby A:

TPSA coefficients

$$T_f^n(X;A) = \sum_{k=0}^{n} \frac{f_A^{(k)}}{k!}(X;A)^k = \sum_{k=0}^{n} \boxed{\frac{1}{k!}} \sum_{\boxed{|\vec{m}|=k}} \boxed{\binom{k}{\vec{m}} \frac{\partial^k f}{\partial X^{\vec{m}}}}\bigg|_A (X;A)^{\vec{m}} \quad \text{with} \quad \boxed{\binom{k}{\vec{m}}} = \frac{k!}{c_1!\,c_2!\ldots c_v!}$$

monomials of order k

multinomial

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA in a nutshell

- Generalised Truncated Power Series Algebra **IPAC 2015**
  - ➡ Multivariate Taylor polynomials of order n in ℝ & ℂ.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

f(x) is an analytic function, $T_f^n(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence h:* $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables *(x,y)* at order 2 nearby *(a,b)*:

$$T_f^2(x,y;a,b) = f(a,b) + \underbrace{\boxed{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)}}_{= f^{(1)}_{(a,b)}(x-a,y-b)} + \ldots$$

$$+\frac{1}{2!}\left(\underbrace{\boxed{\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + \boxed{2\frac{\partial^2 f}{\partial x \partial y}}\,\Big|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2}}_{= f^{(2)}_{(a,b)}(x-a,y-b)}\right)$$

homogeneous polynomials

f must not depend on the integration path, i.e. must derive from a potential!

v variables X at order n nearby A: **TPSA coefficients**

$$T_f^n(X;A) = \sum_{k=0}^{n} \frac{f_A^{(k)}}{k!}(X;A)^k = \sum_{k=0}^{n} \boxed{\frac{1}{k!}} \sum_{\boxed{|\vec{m}|=k}} \boxed{\binom{k}{\vec{m}} \frac{\partial^k f}{\partial X^{\vec{m}}}}\Bigg|_A (X;A)^{\vec{m}} \quad \text{with} \quad \boxed{\binom{k}{\vec{m}}} = \frac{k!}{c_1!\,c_2!\ldots c_v!}$$

monomials of order k **multinomial**

GTPSA in a nutshell

Beams Department

- Generalised Truncated Power Series Algebra **IPAC 2015** **Github MAD**
  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

f(x) is an analytic function, $T_f^n(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence h:* $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables (x,y) at order 2 nearby (a,b):

$$T_f^2(x,y;a,b) = f(a,b) + \underbrace{\boxed{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)}}_{= f^{(1)}_{(a,b)}(x-a,\,y-b)} + \ldots$$

$$+\frac{1}{2!}\underbrace{\boxed{\left(\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + \boxed{2\frac{\partial^2 f}{\partial x \partial y}}\bigg|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2\right)}}_{= f^{(2)}_{(a,b)}(x-a,\,y-b)}$$

homogeneous polynomials

f must not depend on the integration path, i.e. must derive from a potential!

ν variables X at order n nearby A: **TPSA coefficients**

$$T_f^n(X;A) = \sum_{k=0}^{n} \frac{f_A^{(k)}}{k!}(X;A)^k = \sum_{k=0}^{n} \boxed{\frac{1}{k!}} \sum_{\boxed{|\vec{m}|=k}} \boxed{\binom{k}{\vec{m}} \frac{\partial^k f}{\partial X^{\vec{m}}}}\Bigg|_A (X;A)^{\vec{m}} \quad \text{with} \quad \boxed{\binom{k}{\vec{m}}} = \frac{k!}{c_1!\,c_2!\ldots c_\nu!}$$

monomials of order k **multinomial**

- Generalised Truncated Power Series Algebra

IPAC 2015

2017-2018

Github MAD

  - ➡ Multivariate Taylor polynomials of order n in $\mathbb{R}$ & $\mathbb{C}$.
  - ➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f:

$$T^n_f(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R^n_f(x;a) = \lim_{n\to\infty} f(x) - T^n_f(x;a) = 0$$

f(x) is an analytic function, $T^n_f(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence h:* $\min_{h>0}\lim_{n\to\infty} R^n_f(a \pm h; a) \neq 0.$

2 variables (x,y) at order 2 nearby (a,b):

$$= f^{(1)}_{(a,b)}(x-a, y-b)$$

$$T^2_f(x,y;a,b) = f(a,b) + \left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b) + \ldots$$

homogeneous polynomials

f must not depend on the integration path, i.e. must derive from a potential!

$$+\frac{1}{2!}\left(\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + \boxed{2\frac{\partial^2 f}{\partial x \partial y}}\Bigg|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2\right)$$

$$= f^{(2)}_{(a,b)}(x-a, y-b)$$

v variables X at order n nearby A:

TPSA coefficients

$$T^n_f(X;A) = \sum_{k=0}^{n} \frac{f_A^{(k)}}{k!}(X;A)^k = \sum_{k=0}^{n} \boxed{\frac{1}{k!}} \sum_{|\vec{m}|=k} \boxed{\binom{k}{\vec{m}} \frac{\partial^k f}{\partial X^{\vec{m}}}}\Bigg|_A (X;A)^{\vec{m}} \quad \text{with} \quad \boxed{\binom{k}{\vec{m}}} = \frac{k!}{c_1!\,c_2!\ldots c_v!}$$

monomials of order k

multinomial

Accuracy of TPSA (myths and legends)

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  ➡ derivatives are computed using **automatic differentiation** (AD).

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Accuracy of TPSA (myths and legends)

Beams Department

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k$$

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k \,; \quad f(a+h) \approx \sum_{k=0}^{n} \frac{f_a^{(k)}}{k!} h^k$$

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k \;; \qquad f(a+h) \approx \underbrace{\sum_{k=0}^{n} \frac{f_a^{(k)}}{k!} h^k}_{T_f^n(a+h;a)}$$

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k \; ; \quad f(a+h) \approx \underbrace{\sum_{k=0}^{n} \frac{f_a^{(k)}}{k!} h^k}_{T_f^n(a+h;a)}$$

Accuracy of TPSA (myths and legends)

BE Beams Department

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k\,; \quad f(a+h) \approx \underbrace{\sum_{k=0}^{n} \frac{f_a^{(k)}}{k!}h^k}_{T_f^n(a+h;a)}\,; \quad f_{a+h}^{(k)} \approx \frac{\mathrm{d}^k T_f^n(x;a)}{\mathrm{d}x^k}(a+h)$$

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k \,; \quad f(a+h) \approx \underbrace{\sum_{k=0}^{n} \frac{f_a^{(k)}}{k!} h^k}_{T_f^n(a+h;a)} \,; \quad f_{a+h}^{(k)} \approx \frac{\mathrm{d}^k T_f^n(x;a)}{\mathrm{d}x^k}(a+h)$$

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k\,; \quad f(a+h) \approx \underbrace{\sum_{k=0}^{n} \frac{f_a^{(k)}}{k!}h^k}_{T_f^n(a+h;a)}\,; \quad f_{a+h}^{(k)} \approx \frac{\mathrm{d}^k T_f^n(x;a)}{\mathrm{d}x^k}(a+h)$$

order n is constant
order n-1 is linear in h

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k\,; \quad f(a+h) \approx \underbrace{\sum_{k=0}^{n} \frac{f_a^{(k)}}{k!} h^k}_{T_f^n(a+h;a)}\,; \quad f_{a+h}^{(k)} \approx \frac{\mathrm{d}^k T_f^n(x;a)}{\mathrm{d}x^k}(a+h)$$

Matrix codes don't do better!

order n is constant
order n-1 is linear in h

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

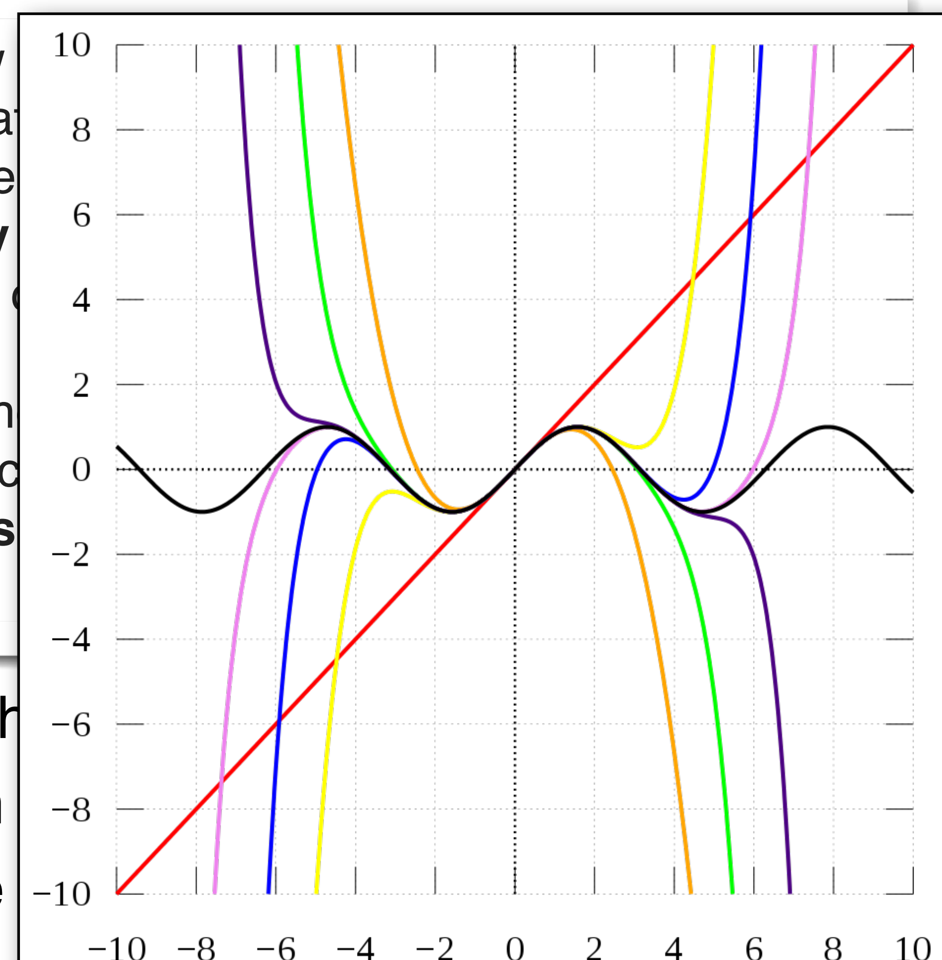
Accuracy of TPSA (myths and legends)

BE Beams Department

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how
> elementary arithmetic operations (addition, subtraction, multiplica
> functions (exp, log, sin, cos, etc.). By applying the chain rule repe
> **of arbitrary order can be computed automatically, accurately**
> most a small constant factor more arithmetic operations than the
>
> Symbolic differentiation can lead to **inefficient code** and faces th
> program into a single expression, while numerical differentiation c
> discretization process and cancellation. **Both classical methods**
> **higher derivatives, where complexity and errors increase.**



sin x and its Taylor approximations nearby 0 by polynomials of degree 1, 3, 5, 7, 9, 11 and 13.

- MAD-NG includes a complete toolbox (i.e. module) to h
  - ➡ users have full access to GTPSA and DAmaps from
  - ➡ users can manipulate DAmaps stored in the MTable
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k \; ; \quad f(a+h) \approx \underbrace{\sum_{k=0}^{n} \frac{f_a^{(k)}}{k!}h^k}_{T_f^n(a+h;a)} \; ; \quad f_{a+h}^{(k)} \approx \frac{\mathrm{d}^k T_f^n(x;a)}{\mathrm{d}x^k}(a+h)$$

Matrix codes don't do better!

order n is constant
order n-1 is linear in h

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Accuracy of TPSA (myths and legends)

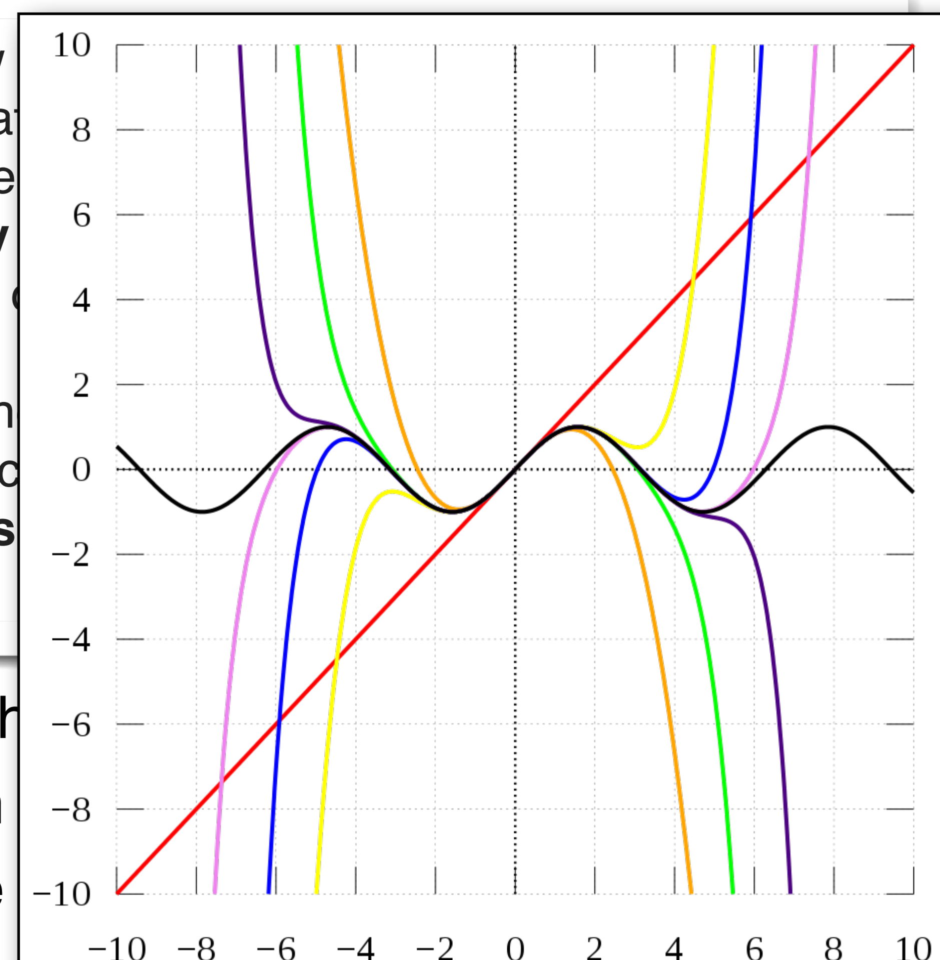
- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ derivatives are computed using **automatic differentiation** (AD).

from Wikipedia

AD exploits the fact that every computer program, no matter how
elementary arithmetic operations (addition, subtraction, multiplica
functions (exp, log, sin, cos, etc.). By ... chain rule repe
of arbitrary order can ... lly, accurately
most ... arithmetic operations than the

Symbolic differentiation can lead to **inefficient code** and faces th
program into a single expression, while numerical differentiation c
discretization process and cancellation. **Both classical methods**
higher derivatives, where complexity and errors increase.

Functions of TPSAs ≠ TPSAs as functions
exact ≠ approximate



sin x and its Taylor approximations nearby 0 by polynomials of degree 1, 3, 5, 7, 9, 11 and 13.

- MAD-NG includes a complete toolbox (i.e. module) to h
  - ➡ users have full access to GTPSA and DAmaps from
  - ➡ users can manipulate DAmaps stored in the MTable
- *So when DAmap/TPSA introduce errors?*
  - ➡ If they are used as *functions* (e.g. evaluated), instead
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k \;; \quad f(a+h) \approx \underbrace{\sum_{k=0}^{n} \frac{f_a^{(k)}}{k!} h^k}_{T_f^n(a+h;a)} \;; \quad f_{a+h}^{(k)} \approx \frac{\mathrm{d}^k T_f^n(x;a)}{\mathrm{d}x^k}(a+h)$$

Matrix codes don't do better!

order n is constant
order n-1 is linear in h

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

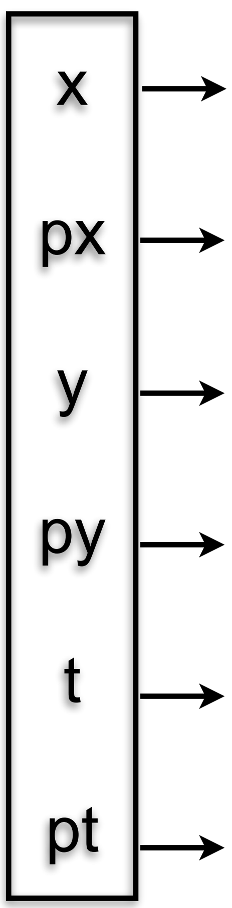
- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)



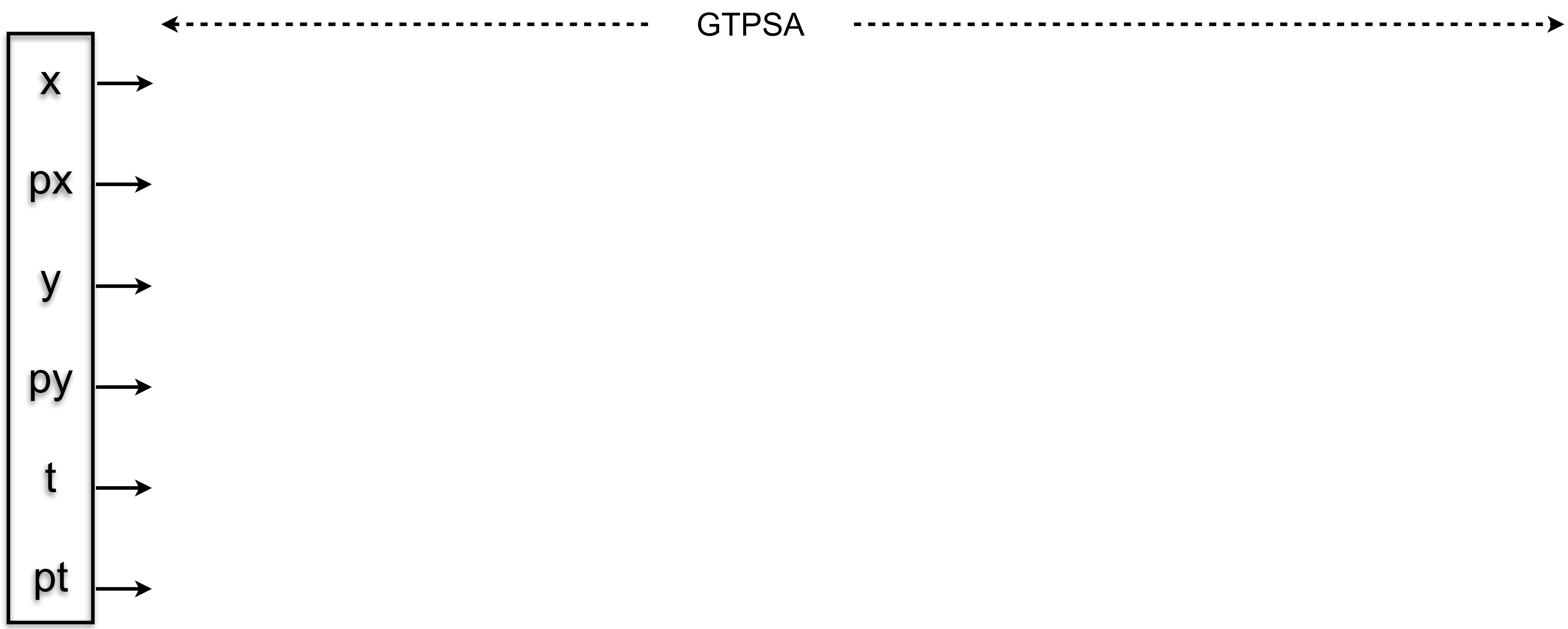
Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

Beams Department

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)

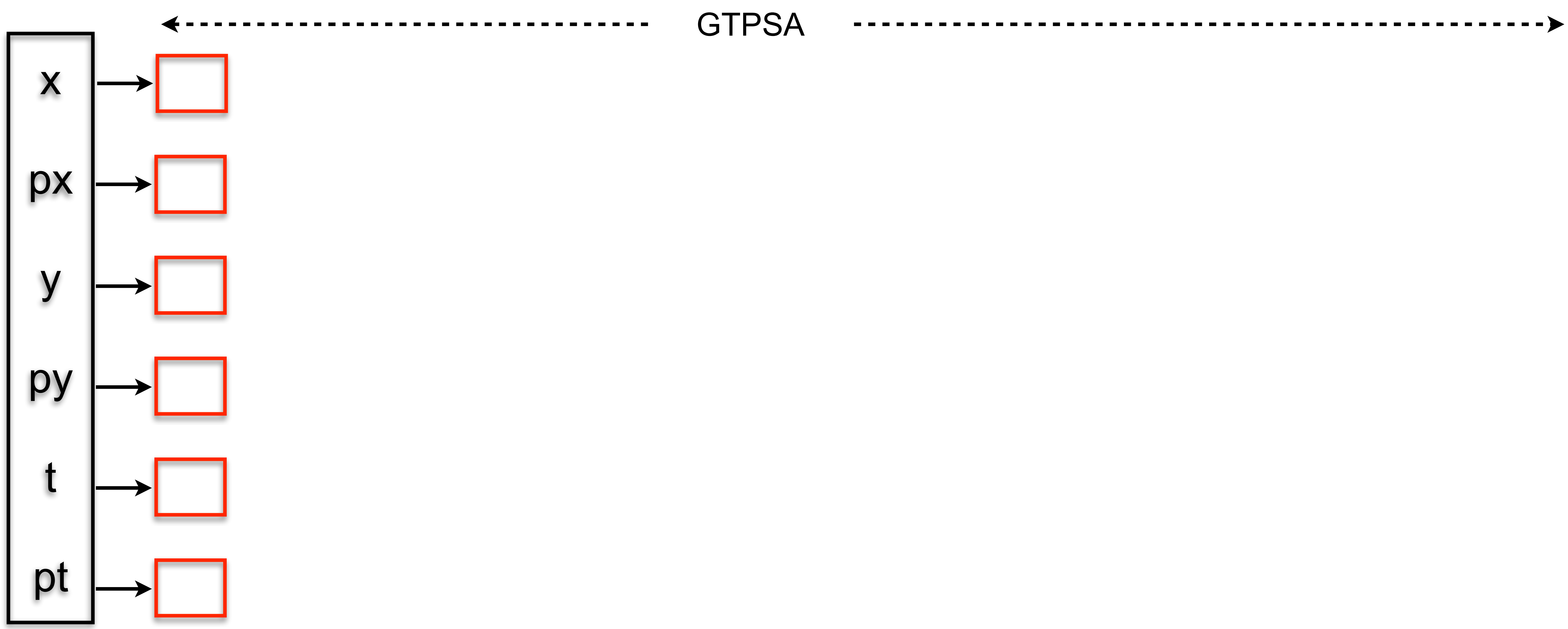


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)

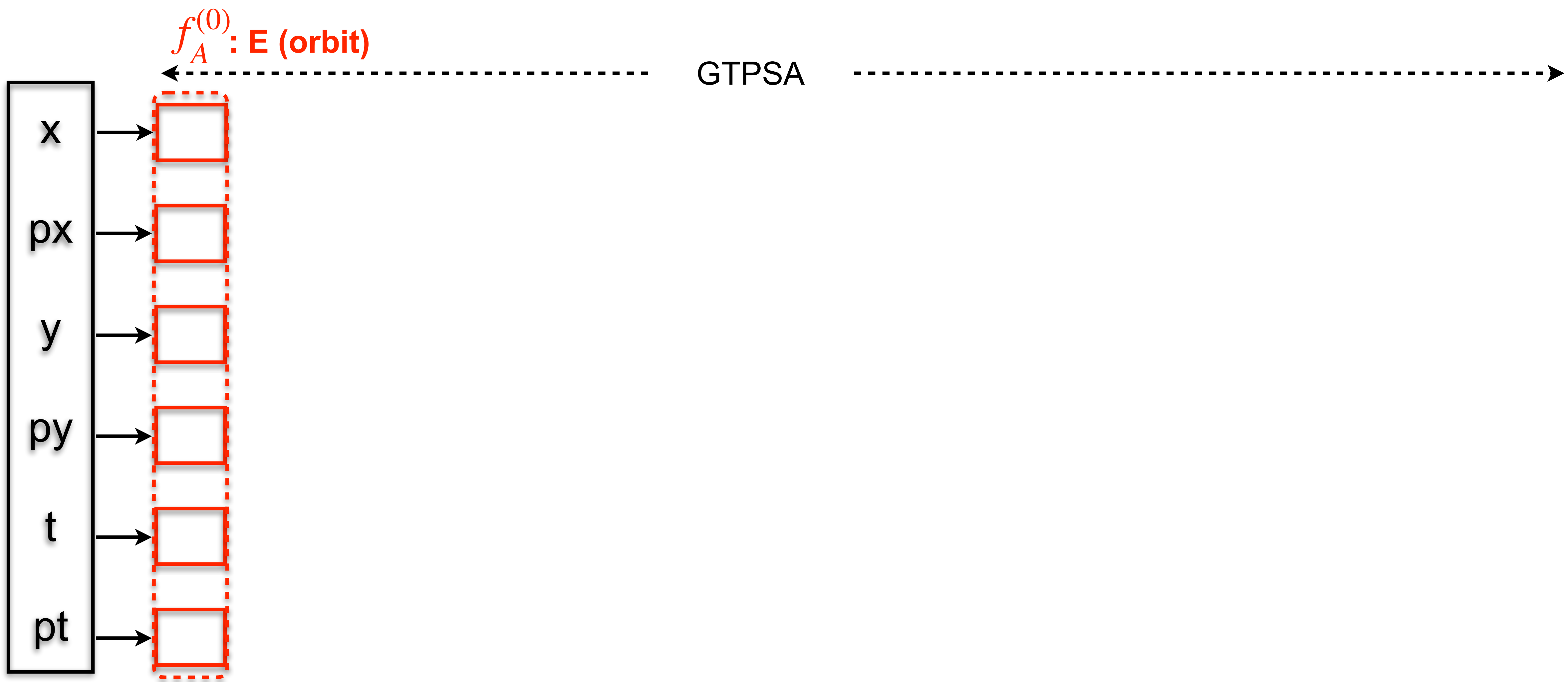


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)

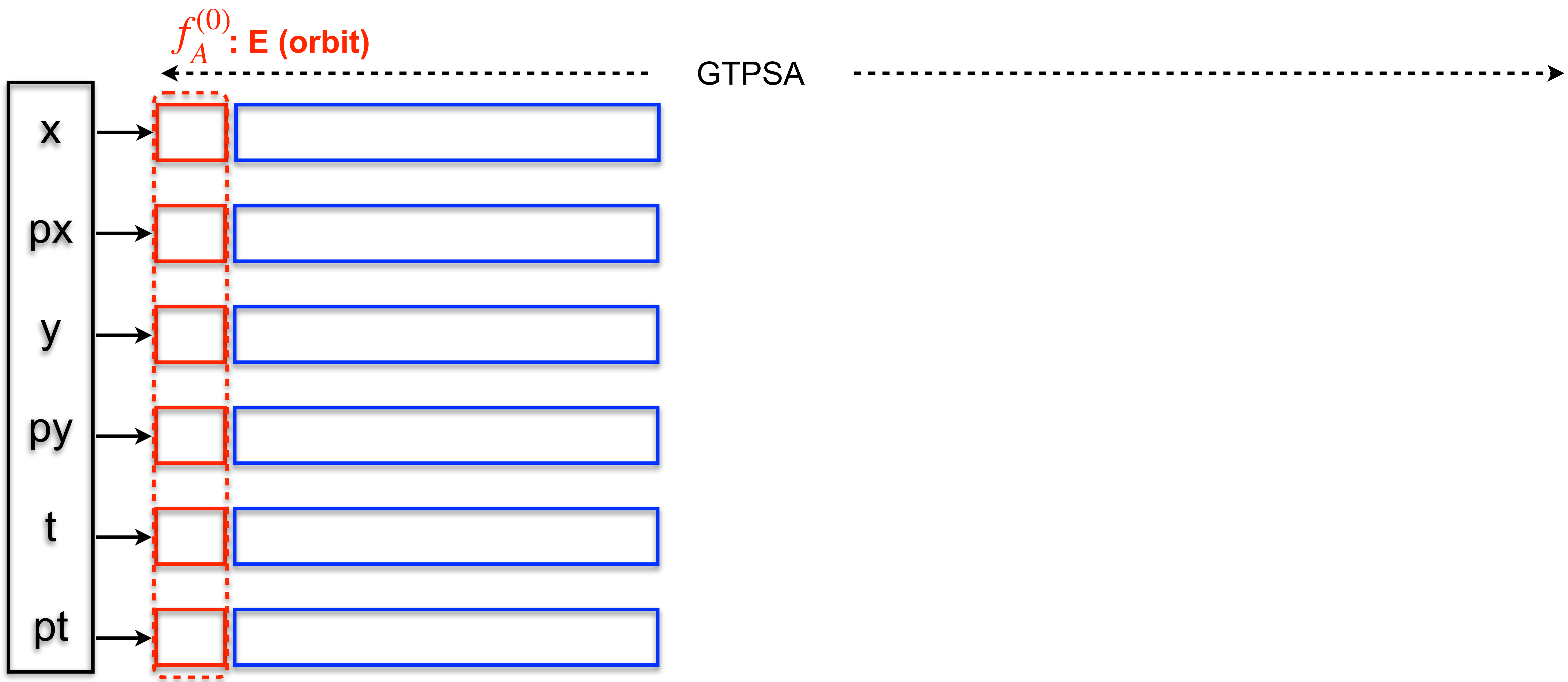


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

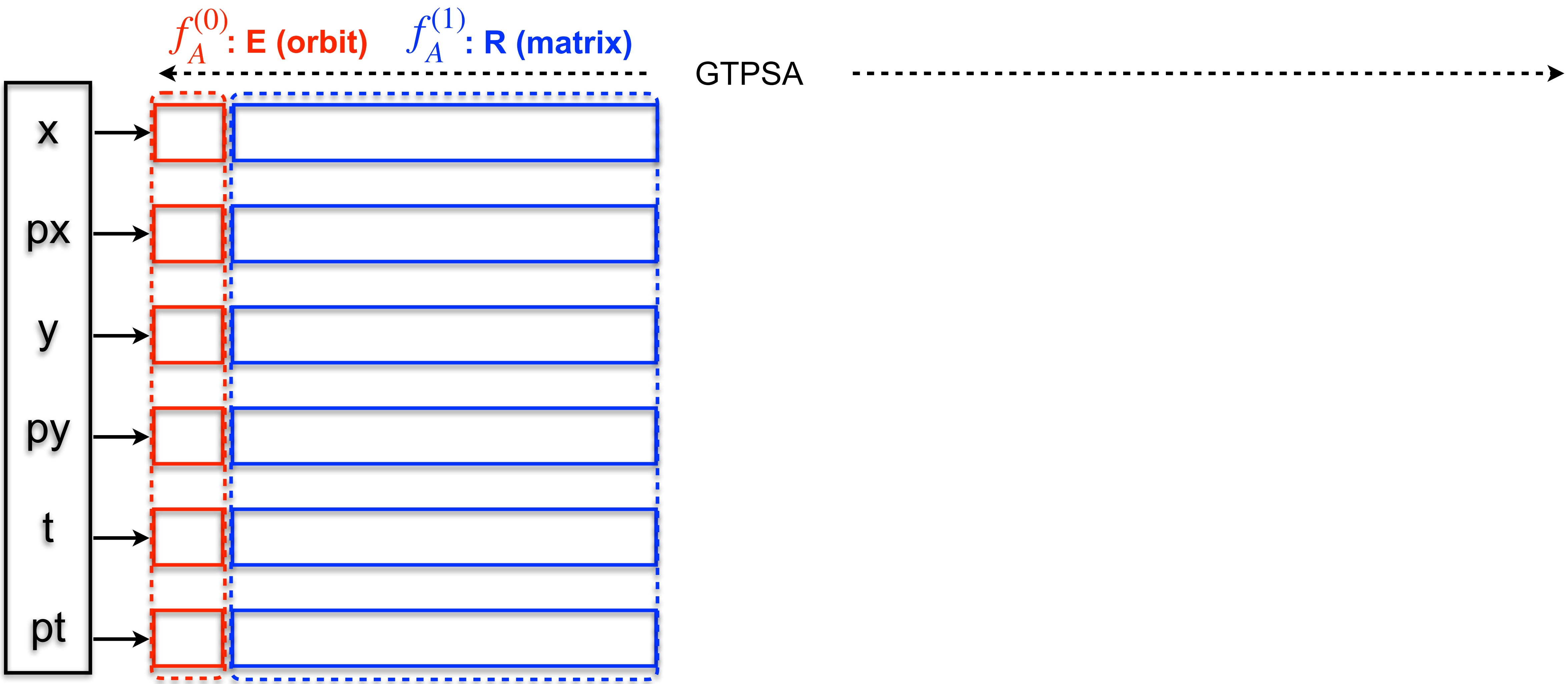
DA map of 6 variables at order 2 (e.g. MAD-X twiss)



DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)

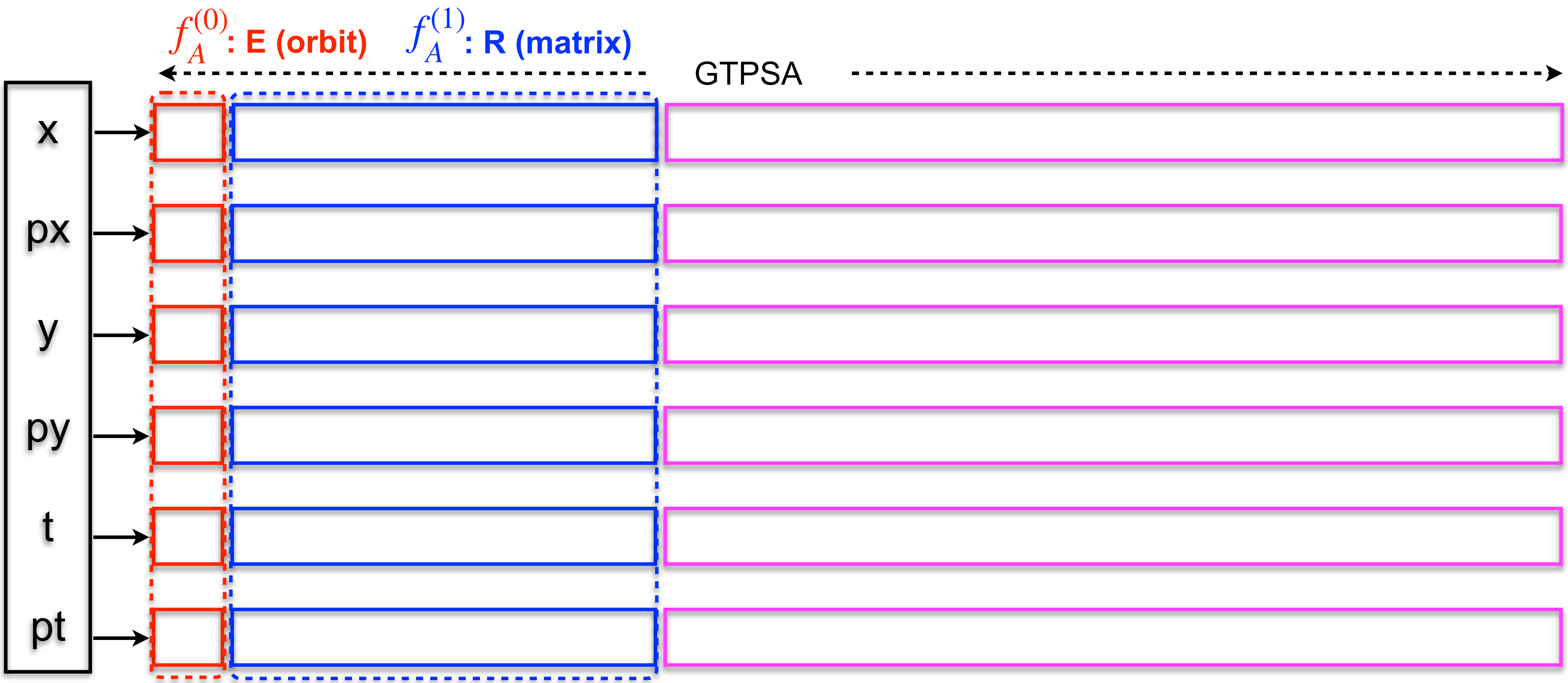


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)

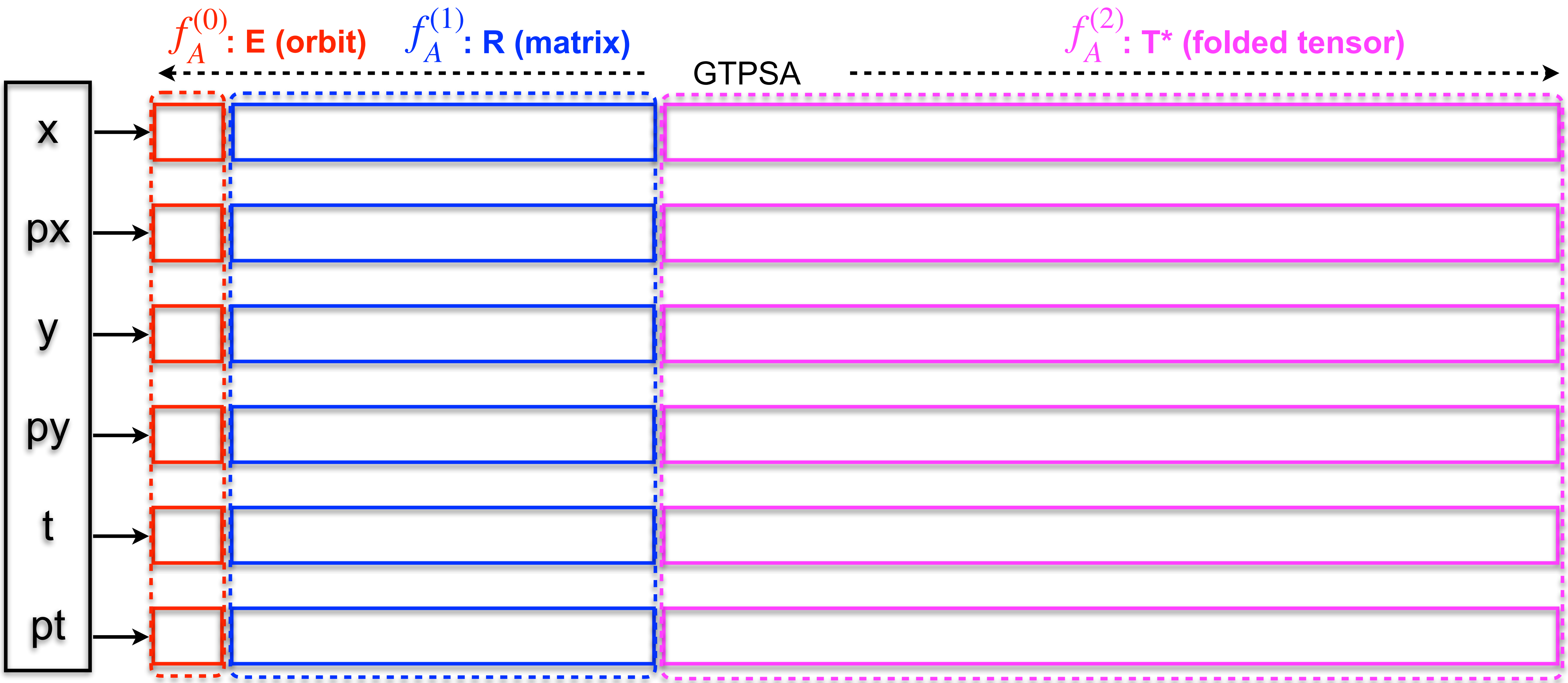


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)

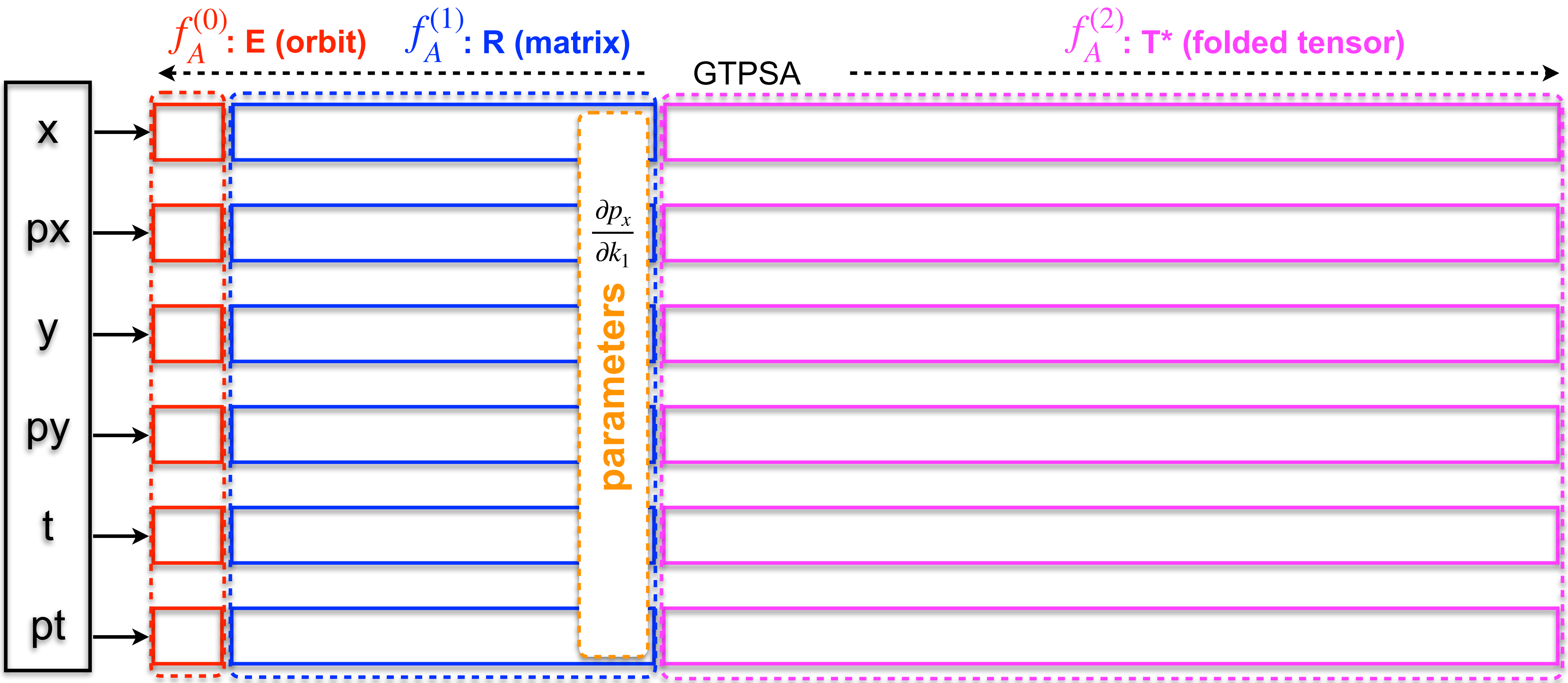


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)

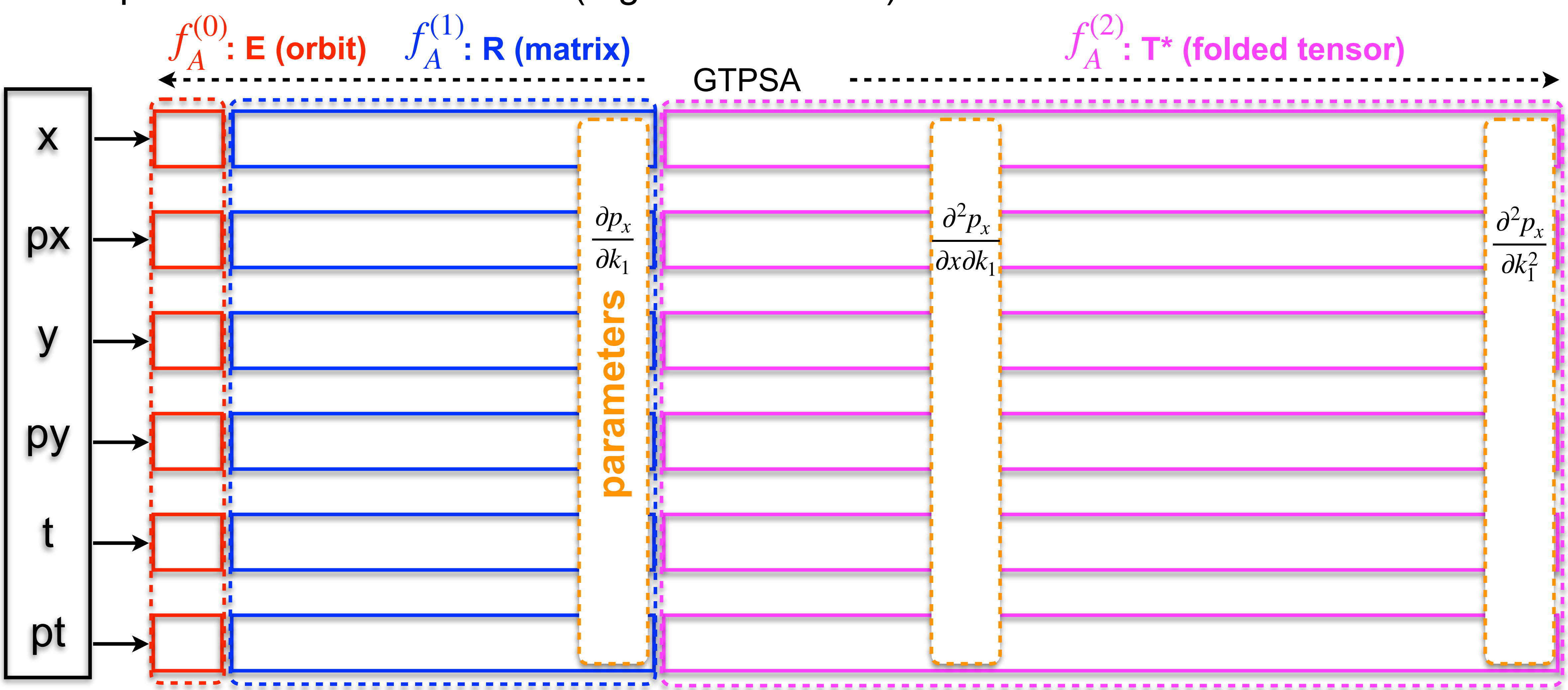


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - Handles user defined parameters.
  - Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)

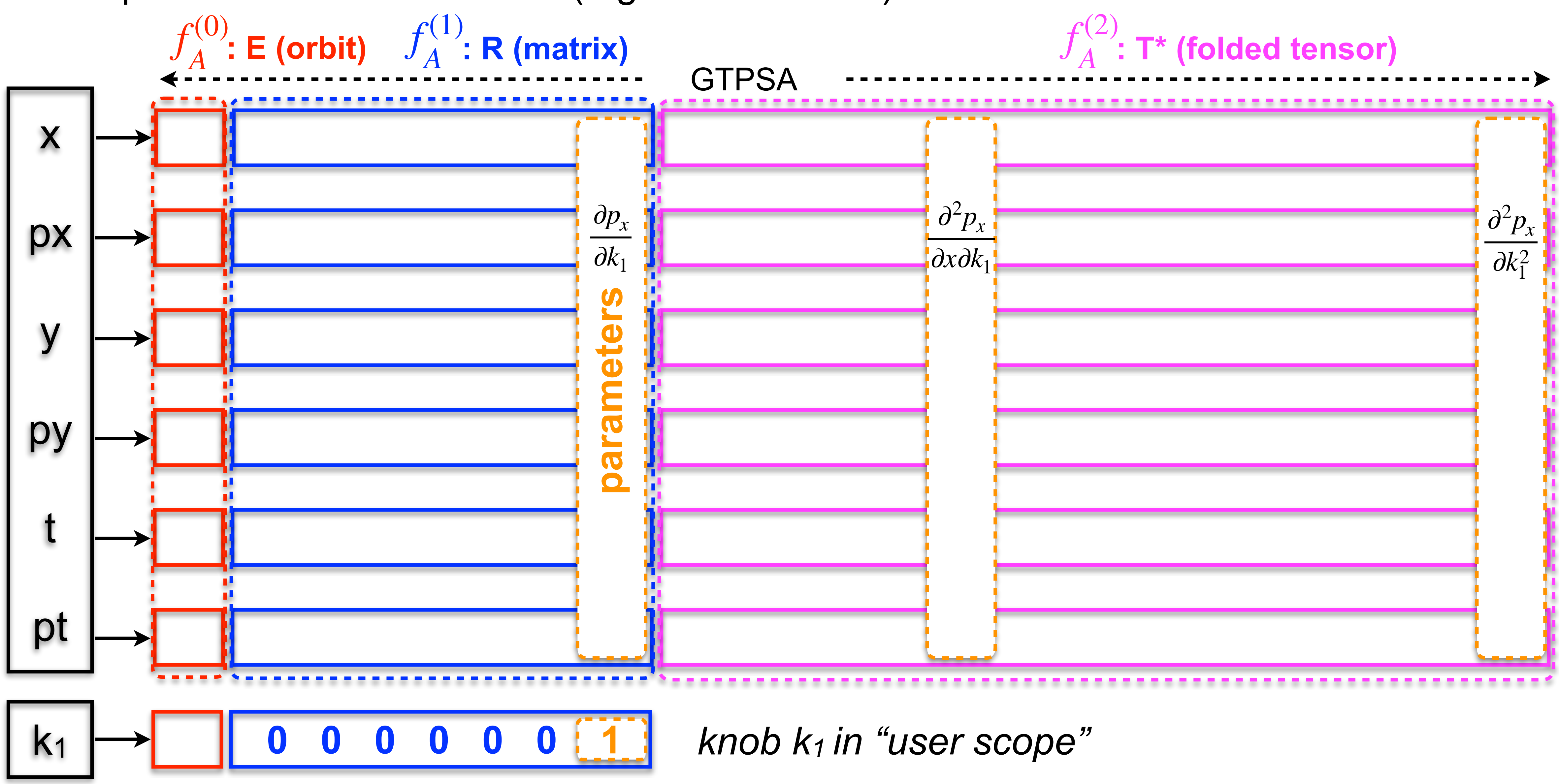


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)

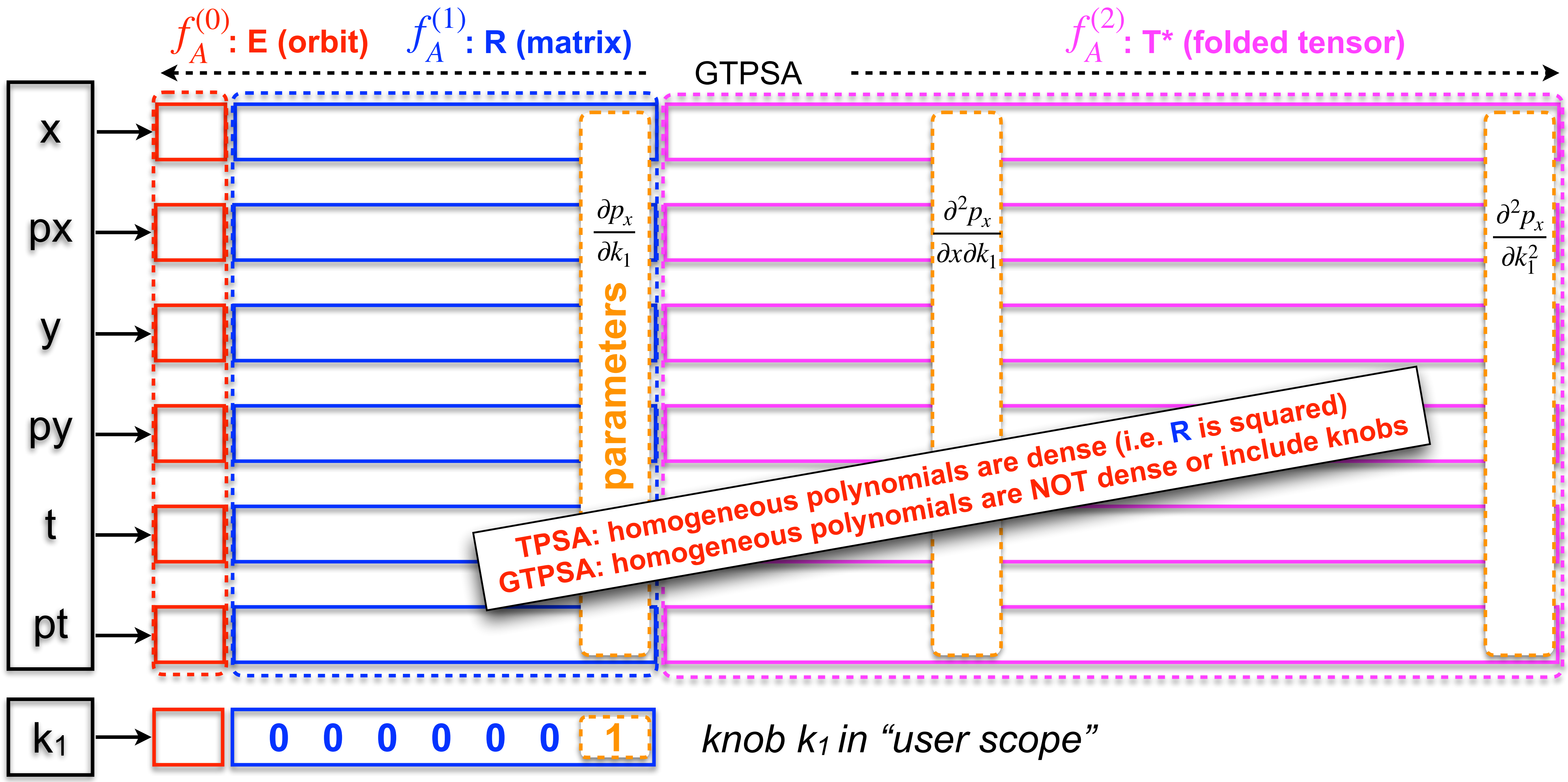


Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DA map

- Differential Algebra maps
  - ➡ Tuple of GTPSA, e.g. 6D phase space uses 6 GTPSA.
  - ➡ Handles user defined parameters.
  - ➡ Behaves like particles for the scalar part (orbit).

DA map of 6 variables at order 2 (e.g. MAD-X twiss)



DAmap vs. Matrix sizes

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DAmap vs. Matrix sizes

TPSA: homogeneous polynomials are dense with $\binom{n+v}{v} = \dfrac{(n+v)!}{n!\,v!}$ coefficients

DAmap vs. Matrix sizes

TPSA: homogeneous polynomials are dense with $\binom{n+v}{v} = \frac{(n+v)!}{n!\,v!}$ coefficients

GTPSA: homogeneous polynomials are NOT dense (no direct formula, only upper bound)

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DAmap vs. Matrix sizes

TPSA: homogeneous polynomials are dense with $\binom{n+v}{v} = \frac{(n+v)!}{n!\,v!}$ coefficients

GTPSA: homogeneous polynomials are NOT dense (no direct formula, only upper bound)

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	12	20	30	42	56	72	90	110	132	156	182
3	12	30	60	105	168	252	360	495	660	858	1092	1365
4	20	60	140	280	504	840	1320	1980	2860	4004	5460	7280
5	30	105	280	630	1260	2310	3960	6435	10010	15015	21840	30940
6	42	**168**	504	1260	2772	5544	10296	**18018**	30030	48048	74256	111384
7	56	252	840	2310	5544	12012	24024	45045	80080	136136	222768	352716
8	72	360	1320	3960	10296	24024	51480	102960	194480	350064	604656	1007760

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

CERN

DAmap vs. Matrix sizes

BE Beams Department

TPSA: homogeneous polynomials are dense with $\binom{n+v}{v} = \dfrac{(n+v)!}{n!\,v!}$ coefficients

GTPSA: homogeneous polynomials are NOT dense (no direct formula, only upper bound)

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	12	20	30	42	56	72	90	110	132	156	182
3	12	30	60	105	168	252	360	495	660	858	1092	1365
4	20	60	140	280	504	840	1320	1980	2860	4004	5460	7280
5	30	105	280	630	1260	2310	3960	6435	10010	15015	21840	30940
6	42	**168**	504	1260	2772	5544	10296	**18018**	30030	48048	74256	111384
7	56	252	840	2310	5544	12012	24024	45045	80080	136136	222768	352716
8	72	360	1320	3960	10296	24024	51480	102960	194480	350064	604656	1007760

DA map: $v\binom{n+v}{v}$

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

CERN

DAmap vs. Matrix sizes

BE Beams Department

TPSA: homogeneous polynomials are dense with $\binom{n+v}{v} = \dfrac{(n+v)!}{n!\,v!}$ coefficients

GTPSA: homogeneous polynomials are NOT dense (no direct formula, only upper bound)

DA map: $v\binom{n+v}{v}$

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	12	20	30	42	56	72	90	110	132	156	182
3	12	30	60	105	168	252	360	495	660	858	1092	1365
4	20	60	140	280	504	840	1320	1980	2860	4004	5460	7280
5	30	105	280	630	1260	2310	3960	6435	10010	15015	21840	30940
6	42	**168**	504	1260	2772	5544	10296	**18018**	30030	48048	74256	111384
7	56	252	840	2310	5544	12012	24024	45045	80080	136136	222768	352716
8	72	360	1320	3960	10296	24024	51480	102960	194480	350064	604656	1007760

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	14	30	62	126	254	510	1022	2046	4094	8190	16382
3	12	39	120	363	1092	3279	9840	29523	88572	265719	797160	2391483
4	20	84	340	1364	5460	21844	87380	349524	1398100	5592404	22369620	89478484
5	30	155	780	3905	19530	97655	488280	2441405	12207030	61035155	305175780	1525878905
6	42	**258**	1554	9330	55986	335922	2015538	**12093234**	72559410	435356466	2612138802	15672832818
7	56	399	2800	19607	137256	960799	6725600	47079207	329554456	2306881199	16148168400	113037178807
8	72	584	4680	37448	299592	2396744	19173960	153391688	1227133512	9817068104	78536544840	628292358728

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DAmap vs. Matrix sizes

TPSA: homogeneous polynomials are dense with $\binom{n+v}{v} = \dfrac{(n+v)!}{n!\,v!}$ coefficients

GTPSA: homogeneous polynomials are NOT dense (no direct formula, only upper bound)

DA map: $v\binom{n+v}{v}$

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	12	20	30	42	56	72	90	110	132	156	182
3	12	30	60	105	168	252	360	495	660	858	1092	1365
4	20	60	140	280	504	840	1320	1980	2860	4004	5460	7280
5	30	105	280	630	1260	2310	3960	6435	10010	15015	21840	30940
6	42	168	504	1260	2772	5544	10296	18018	30030	48048	74256	111384
7	56	252	840	2310	5544	12012	24024	45045	80080	136136	222768	352716
8	72	360	1320	3960	10296	24024	51480	102960	194480	350064	604656	1007760

Matrix: $\sum_{k=0}^{n} v^{k+1} = \dfrac{v(v^{n+1}-1)}{v-1}$

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	14	30	62	126	254	510	1022	2046	4094	8190	16382
3	12	39	120	363	1092	3279	9840	29523	88572	265719	797160	2391483
4	20	84	340	1364	5460	21844	87380	349524	1398100	5592404	22369620	89478484
5	30	155	780	3905	19530	97655	488280	2441405	12207030	61035155	305175780	1525878905
6	42	258	1554	9330	55986	335922	2015538	12093234	72559410	435356466	2612138802	15672832818
7	56	399	2800	19607	137256	960799	6725600	47079207	329554456	2306881199	16148168400	113037178807
8	72	584	4680	37448	299592	2396744	19173960	153391688	1227133512	9817068104	78536544840	628292358728

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DAmap vs. Matrix sizes

TPSA: homogeneous polynomials are dense with $\binom{n+v}{v} = \frac{(n+v)!}{n!\,v!}$ coefficients

GTPSA: homogeneous polynomials are NOT dense (no direct formula, only upper bound)

DA map: $v\binom{n+v}{v}$

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	12	20	30	42	56	72	90	110	132	156	182
3	12	30	60	105	168	252	360	495	660	858	1092	1365
4	20	60	140	280	504	840	1320	1980	2860	4004	5460	7280
5	30	105	280	630	1260	2310	3960	6435	10010	15015	21840	30940
6	42	168	504	1260	2772	5544	10296	18018	30030	48048	74256	111384
7	56	252	840	2310	5544	12012	24024	45045	80080	136136	222768	352716
8	72	360	1320	3960	10296	24024	51480	102960	194480	350064	604656	1007760

Matrix: $\sum_{k=0}^{n} v^{k+1} = \frac{v(v^{n+1}-1)}{v-1}$

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	14	30	62	126	254	510	1022	2046	4094	8190	16382
3	12	39	120	363	1092	3279	9840	29523	88572	265719	797160	2391483
4	20	84	340	1364	5460	21844	87380	349524	1398100	5592404	22369620	89478484
5	30	155	780	3905	19530	97655	488280	2441405	12207030	61035155	305175780	1525878905
6	42	258	1554	9330	55986	335922	2015538	12093234	72559410	435356466	2612138802	15672832818
7	56	399	2800	19607	137256	960799	6725600	47079207	329554456	2306881199	16148168400	113037178807
8	72	584	4680	37448	299592	2396744	19173960	153391688	1227133512	9817068104	78536544840	628292358728

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

BE Beams Department

DAmap vs. Matrix sizes

TPSA: homogeneous polynomials are dense with $\binom{n+v}{v} = \frac{(n+v)!}{n!\,v!}$ coefficients

GTPSA: homogeneous polynomials are NOT dense (no direct formula, only upper bound)

DA map: $v\binom{n+v}{v}$

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	12	20	30	42	56	72	90	110	132	156	182
3	12	30	60	105	168	252	360	495	660	858	1092	1365
4	20	60	140	280	504	840	1320	1980	2860	4004	5460	7280
5	30	105	280	630	1260	2310	3960	6435	10010	15015	21840	30940
6	42	168	504	1260	2772	5544	10296	18018	30030	48048	74256	111384
7	56	252	840	2310	5544	12012	24024	45045	80080	136136	222768	352716
8	72	360	1320	3960	10296	24024	51480	102960	194480	350064	604656	1007760

Matrix: $\sum_{k=0}^{n} v^{k+1} = \frac{v(v^{n+1}-1)}{v-1}$

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	14	30	62	126	254	510	1022	2046	4094	8190	16382
3	12	39	120	363	1092	3279	9840	29523	88572	265719	797160	2391483
4	20	84	340	1364	5460	21844	87380	349524	1398100	5592404	22369620	89478484
5	30	155	780	3905	19530	97655	488280	2441405	12207030	61035155	305175780	1525878905
6	42	258	1554	9330	55986	335922	2015538	12093234	72559410	435356466	2612138802	15672832818
7	56	399	2800	19607	137256	960799	6725600	47079207	329554456	2306881199	16148168400	113037178807
8	72	584	4680	37448	299592	2396744	19173960	153391688	1227133512	9817068104	78536544840	628292358728

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DAmap vs. Matrix sizes

TPSA: homogeneous polynomials are dense with $\binom{n+v}{v} = \frac{(n+v)!}{n!\,v!}$ coefficients

GTPSA: homogeneous polynomials are NOT dense (no direct formula, only upper bound)

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	12	20	30	42	56	72	90	110	132	156	182
3	12	30	60	105	168	252	360	495	660	858	1092	1365
4	20	60	140	280	504	840	1320	1980	2860	4004	5460	7280
5	30	105	280	630	1260	2310	3960	6435	10010	15015	21840	30940
6	42	**168**	504	1260	2772	5544	10296	**18018**	30030	48048	74256	111384
7	56	252	840	2310	5544	12012	24024	45045	80080	136136	222768	352716
8	72	360	1320	3960	10296	24024	51480	102960	194480	350064	604656	1007760

DA map: $v\binom{n+v}{v}$

TPSA manipulate only numbers!

Matrix: $\sum_{k=0}^{n} v^{k+1} = \frac{v(v^{n+1}-1)}{v-1}$

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	14	30	62	126	254	510	1022	2046	4094	8190	16382
3	12	39	120	363	1092	3279	9840	29523	88572	265719	797160	2391483
4	20	84	340	1364	5460	21844	87380	349524	1398100	5592404	22369620	89478484
5	30	155	780	3905	19530	97655	488280	2441405	12207030	61035155	305175780	1525878905
6	42	**258**	1554	9330	55986	335922	2015538	**12093234**	72559410	435356466	2612138802	15672832818
7	56	399	2800	19607	137256	960799	6725600	47079207	329554456	2306881199	16148168400	113037178807
8	72	584	4680	37448	299592	2396744	19173960	153391688	1227133512	9817068104	78536544840	628292358728

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

DAmap vs. Matrix sizes

BE Beams Department

TPSA: homogeneous polynomials are dense with $\binom{n+v}{v} = \frac{(n+v)!}{n!\,v!}$ coefficients

GTPSA: homogeneous polynomials are NOT dense (no direct formula, only upper bound)

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	12	20	30	42	56	72	90	110	132	156	182
3	12	30	60	105	168	252	360	495	660	858	1092	1365
4	20	60	140	280	504	840	1320	1980	2860	4004	5460	7280
5	30	105	280	630	1260	2310	3960	6435	10010	15015	21840	30940
6	42	168	504	1260	2772	5544	10296	18018	30030	48048	74256	111384
7	56	252	840	2310	5544	12012	24024	45045	80080	136136	222768	352716
8	72	360	1320	3960	10296	24024	51480	102960	194480	350064	604656	1007760

DA map: $v\binom{n+v}{v}$

TPSA manipulate only numbers!

TPSA are the only suitable solutions for high orders!

Matrix: $\sum_{k=0}^{n} v^{k+1} = \frac{v(v^{n+1}-1)}{v-1}$

v \ n	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13
2	6	14	30	62	126	254	510	1022	2046	4094	8190	16382
3	12	39	120	363	1092	3279	9840	29523	88572	265719	797160	2391483
4	20	84	340	1364	5460	21844	87380	349524	1398100	5592404	22369620	89478484
5	30	155	780	3905	19530	97655	488280	2441405	12207030	61035155	305175780	1525878905
6	42	258	1554	9330	55986	335922	2015538	12093234	72559410	435356466	2612138802	15672832818
7	56	399	2800	19607	137256	960799	6725600	47079207	329554456	2306881199	16148168400	113037178807
8	72	584	4680	37448	299592	2396744	19173960	153391688	1227133512	9817068104	78536544840	628292358728

GTPSA performance (vs. Berz and Yang)

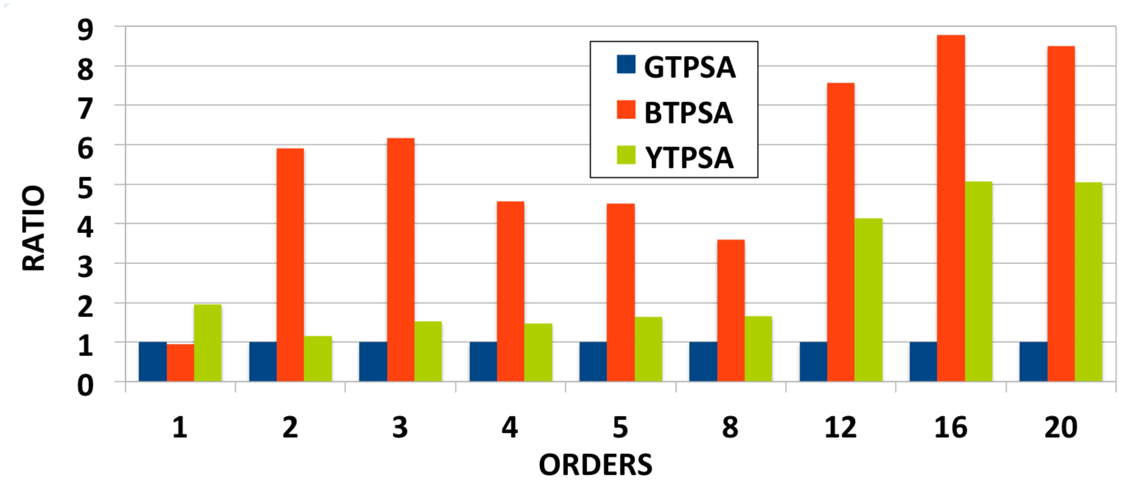


Fig. 5: Relative performance of the multiplications.

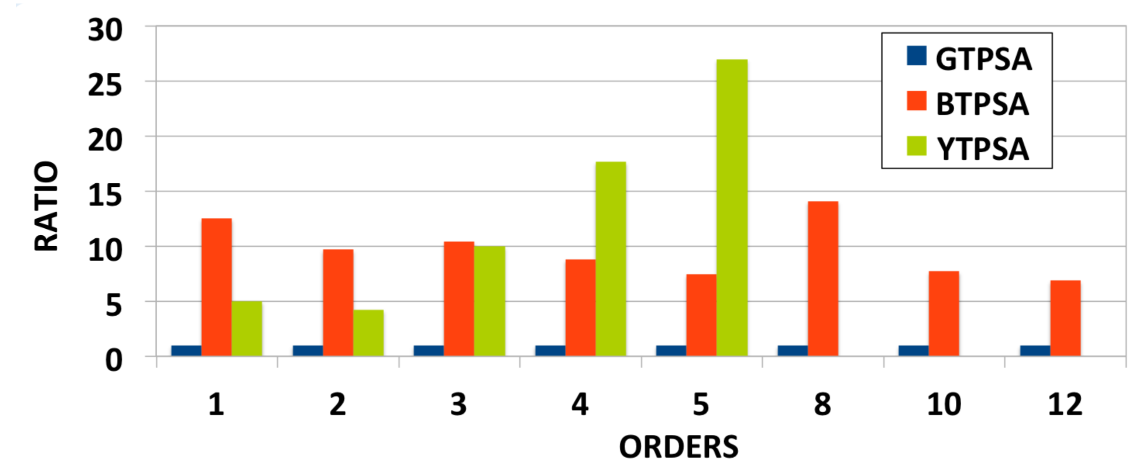


Fig. 6: Relative performance of the compositions.

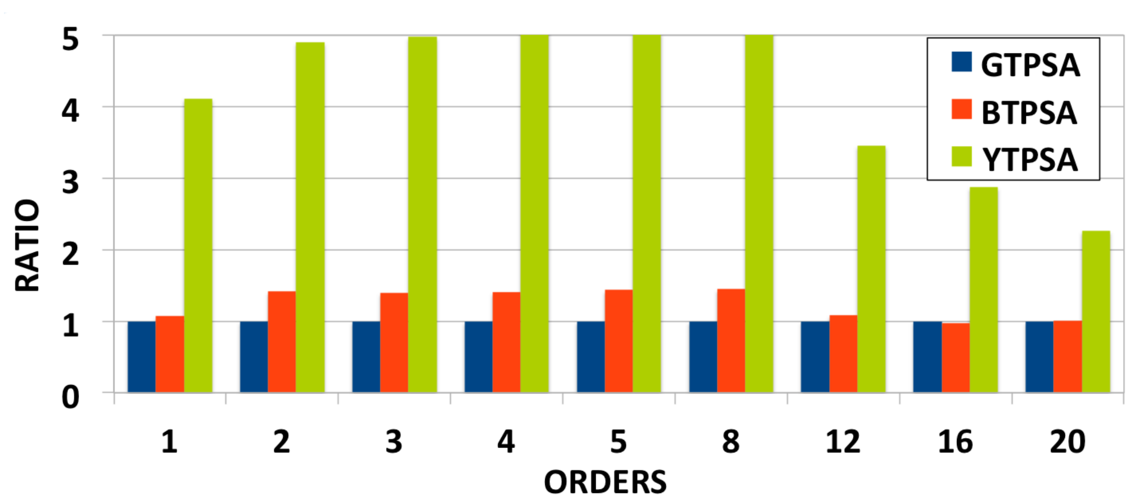


Fig. 2: Relative performance of indexing functions.

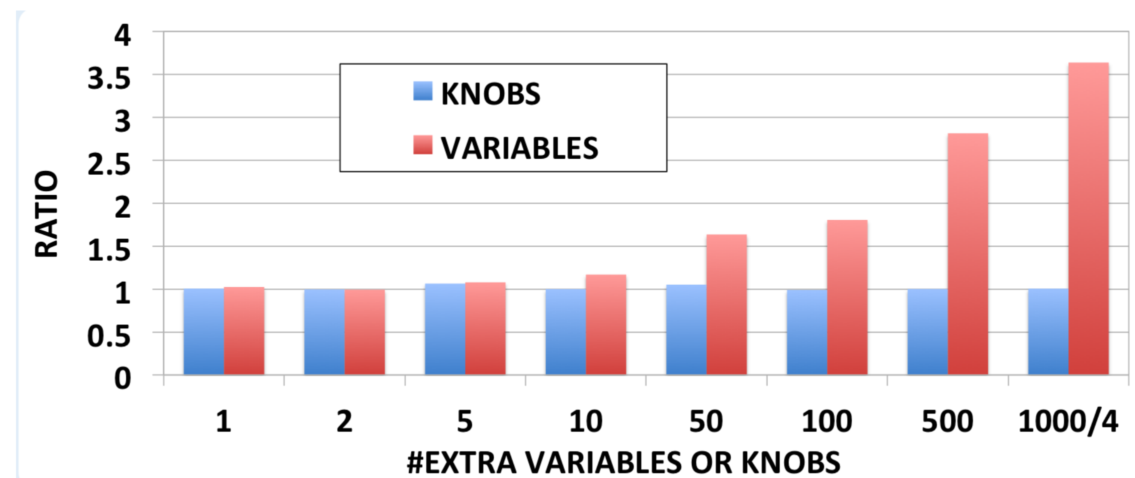


Fig. 4: Relative performance of multiplication at order 2 when using GTPSA with 6 variables and many knobs vs. homogeneous TPSA.

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

GTPSA performance (vs. Berz and Yang)

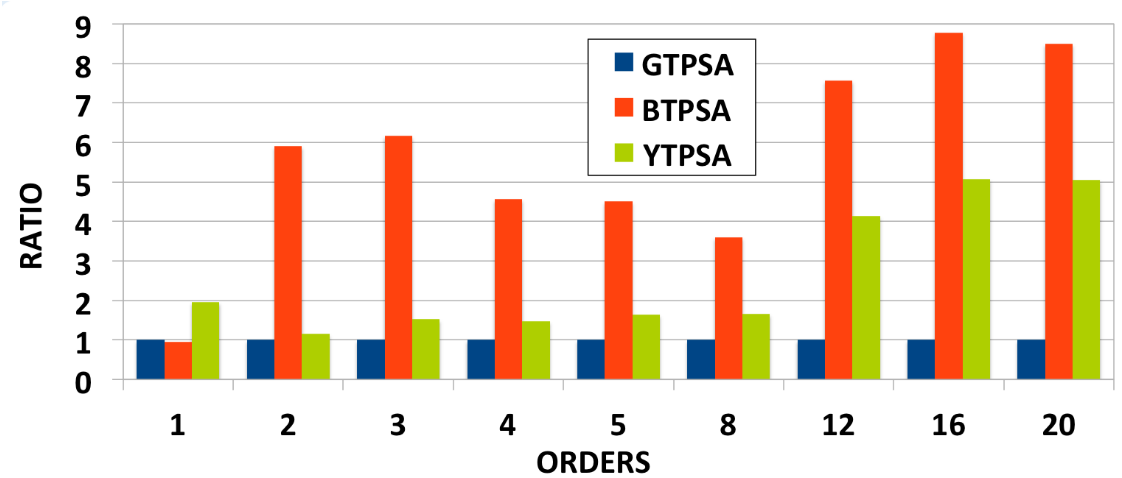


Fig. 5: Relative performance of the multiplications.

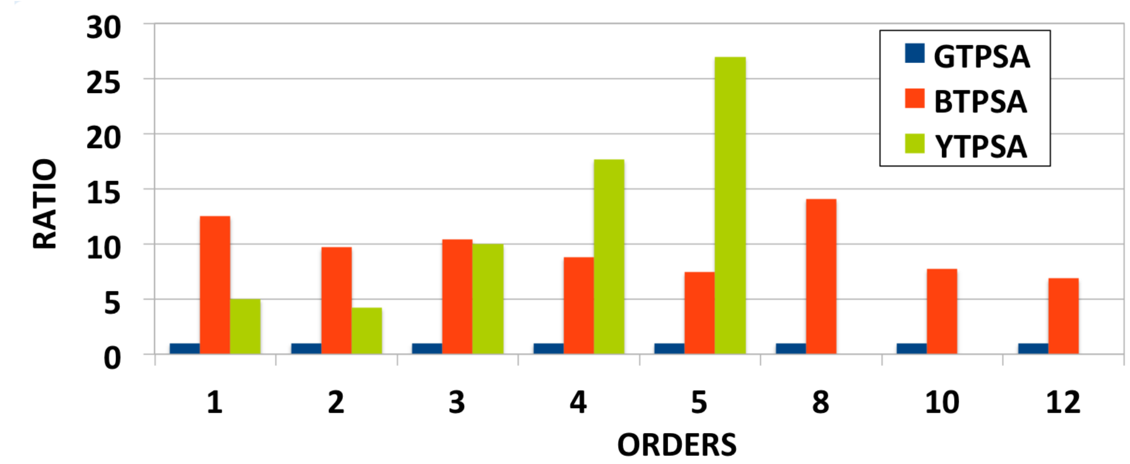


Fig. 6: Relative performance of the compositions.

The smaller the better!

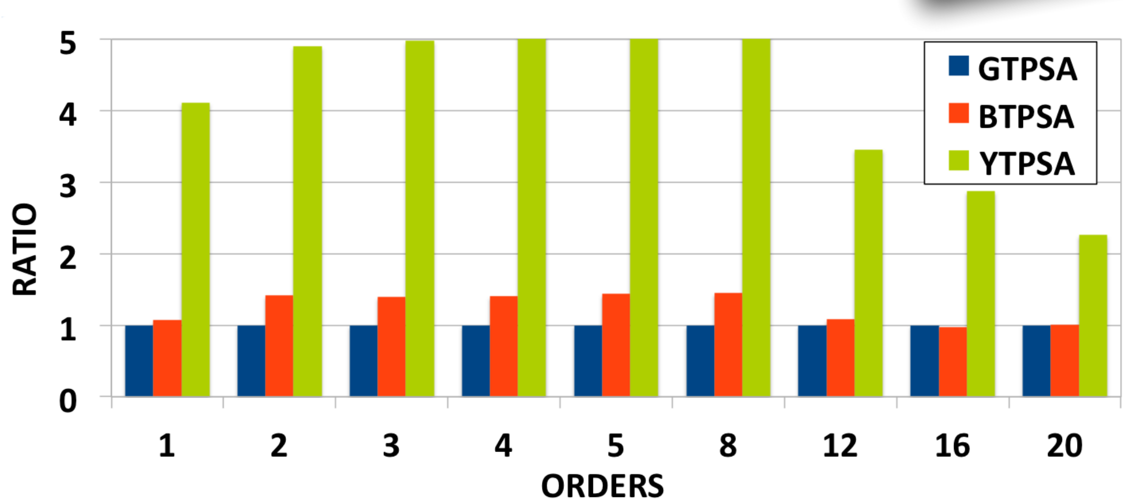


Fig. 2: Relative performance of indexing functions.

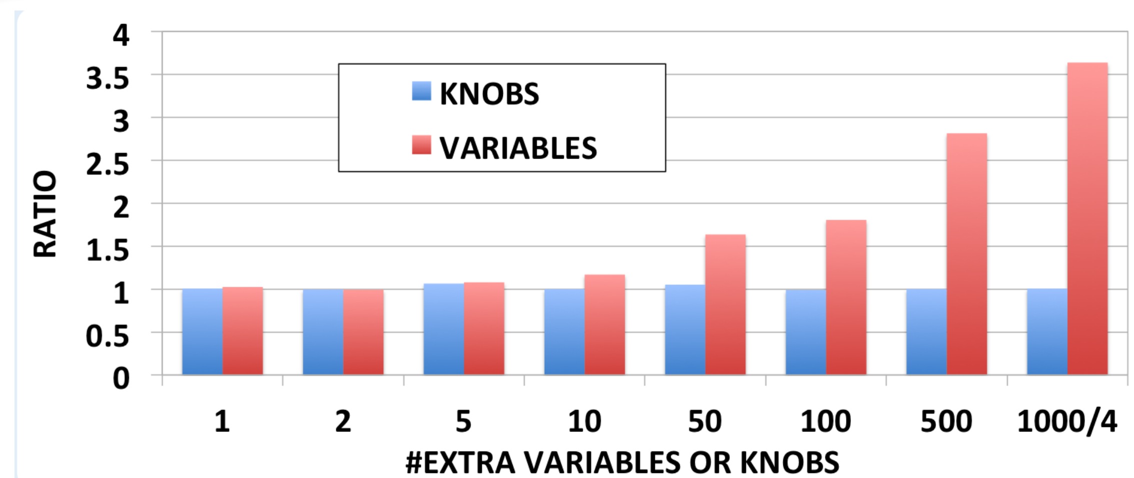


Fig. 4: Relative performance of multiplication at order 2 when using GTPSA with 6 variables and many knobs vs. homogeneous TPSA.

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch